박 사 학 위 논 문

Doctoral Thesis

높은 데이터 유효성을 보장하기 위한 행동-인지 피어-투-피어 프로토콜

Behavior-aware P2P Protocol for High Data Availability

김 경백 (金 俓佰 Kim, Kyungbaek)

전자전산학과 전기및전자공학전공

School of Electrical Engineering & Computer Science

Division of Electrical Engineering

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

2007

높은 데이터 유효성을 보장하기 위한 행동-인지 피어-투-피어 프로토콜

Behavior-aware P2P Protocol for High Data Availability

Behavior-aware P2P Protocol for High Data Availability

Advisor : Professor Daeyeon Park

by

Kim, Kyungbaek

School of Electrical Engineering & Computer Science
Division of Electrical Engineering
Korea Advanced Institute of Science and Technology

A thesis submitted to the faculty of the Korea Advanced Institute of Science and Technology in partial fullfillment of the requirements for the degree of Doctor of Philosophy in the School of Electrical Engineering & Computer Science, Division of Electrical Engineering

Daejeon, Korea
2006. 11. 22.
Approved by

Professor Daeyeon Park
Major Advisor

높은 데이터 유효성을 보장하기 위한 행동-인지 피어-투-피어 프로토콜

김 경백

위 논문은 한국과학기술원 박사학위논문으로 학위논문심사 위원회에서 심사 통과하였음.

2006년 11월 22일

심사위원장	박 대 연	(인)
심사위원	박 규 호	(인)
심사위원	성 단 근	(인)
심사위원	윤 현 수	(인)
심사위원	이 광 형	(인)

DEE 20015030 김 경백. Kim Kyungbaek. Behavior-aware P2P Protocol for High Data Availability. 높은 데이터 유효성을 보장하기 위한 행동-인지 피어-투-피어 프로토콜. School of Electrical Engineering & Computer Science, Division of Electrical Engineering . 2007. 74p. Advisor Prof. Daeyeon Park. Text in English.

Abstract

A lot of research papers discussed the Distributed Hash Table (DHT) based p2p systems to promise that idle resources may be efficiently harvested. However, p2p systems are composed of components with extremely heterogeneous availabilities and to handle churn, the system will generate the heavy information maintenance traffic to keep the efficiency of the DHT. Moreover, the previous researches concentrate on the efficient lookup of the p2p system and they fails to keep high data availability effectively.

This dissertation presents the behavior-aware p2p protocol for high data availability. By using this protocol, the p2p system can reduce the overhead by exploiting the heterogeneous behavior of participant nodes efficiently. Unlike the DHT based p2p which use the static nodeID which is obtained by well-balanced hashing function, the behavior-aware p2p ignores this static nodeID and uses the dynamic nodeID. This dynamic nodeID is composed of the Load Balanced ID which balance the loads of the reliable and powerful nodes and the Load Free ID which reduce the responsibility of normal and weak nodes and eliminate the compulsory maintenance overhead when the churn of them occurs. The assignment of nodeID performs without any central management server. The new nodeID is assigned according to the current network state and the node which processes the join message. After assigning the new nodeID, this nodeID of a node changes on the fly according to its behavior on the p2p system. Finally, every node gets its proper nodeID with its characteristics and each nodes takes the different responsibility in accordance with its nodeID to support p2p system effectively. The reliable and powerful nodes mainly guarantee the data availability and the correctness of lookup. The normal and weak nodes assist them and the churn of them can not harm the data availability and the efficiency of the lookup.

I examine the efficiency of behavior-aware p2p via a event driven simulation. The results show that the behavior-aware p2p keep high data availability with less information maintenance traffic than the DHT based p2p and the routing process is also efficient.

Contents

List of Tables

List of Figures

1. Introduction

In these days, peer-to-peer systems have become an extremely popular platform for large-scale content sharing. Unlike client/server model based storage systems, which centralized the management of data in a few highly reliable servers, peer-to-peer storage systems distribute the burden of data storage and communications among tens of thousands of clients. The wide-spread attraction of this model arises from the promise that idle resources may be efficiently harvested to provide scalable storage services. A lot of research papers discussed the Distributed Hash Table (DHT) based p2p routing algorithms (Chord, Pastry, Tapestry and CAN) [5][6][7][8].

In contrast to traditional systems, peer-to-peer systems are composed of components with extremely heterogeneous availabilities - individually administered host PC's may be turned on and off, join and leave the system, have intermittent connectivity, and are constructed from low-cost low reliability components. For example, one recent study[11] of a popular peer-to-peer file sharing system found that the majority of peers had application-level availability rates of under 20 percent and only 20 percent nodes have server-like profiles. In such an environment, failure is no longer an exceptional event, but is a pervasive condition. At any point in time the majority of hosts in the system are unavailable and those hosts that are available may soon stop servicing requests.

A big issue in current DHT based p2p systems is the high overhead of maintaining DHT routing data structure and the stored data. When a node joins/leaves the system, the affected routing data structure on some existing nodes must be updated accordingly to reflect the change. Moreover, most p2p systems employ some form of the data redundancy to cope with failure and when the membership of nodes changes, these systems generate huge overhead of compulsory copies for the data availability [9][10][1]. Especially, for nodes which join/leave the systems frequently, the p2p system will generate a lot of routing information update traffic and data copy traffic. It does not only increase the consumption of the network bandwidth, but also affects the efficiency of DHT based routing algorithms. Until now, DHT based p2p systems are not widely used in commercial systems yet, most p2p file sharing systems are still using non structured p2p mechanisms.

In this dissertation, I suggest the behavior-aware p2p protocol to keep the high data availability with small maintenance cost by exploiting the heterogeneity of participant nodes efficiently. The node heterogeneity makes the maintenance overhead heavy, but it also gives us the chance to improve the performance, that is, the more reliable and more powerful nodes can handle much more jobs than normal nodes. If the reliable nodes are sorted out from the all participant nodes during the system running, the p2p system can get the chance to use them efficiently and reduce the management overhead. In this case, using the static nodeID which is used by the previous DHT based p2p is hard to change the position and role of each participant nodes in the p2p system. Unlike the DHT based p2p, in the behavior-aware p2p, the nodeID of a node changes on the fly according to its behavior on the p2p system to support the p2p system efficiency and each nodes takes the different responsibility in accordance with its nodeID.

This dynamic nodeID consists of the *Load-Balanced ID (LBID)* and the *Load-Free ID (LFID)*. The participant nodes are classified into the two types, the representative nodes and the transient nodes, according to their nodeIDs which are changed by the characteristics of nodes. The representative nodes are more reliable and more powerful nodes and they act as more important roles such as routing and replication. According to this characteristic, they are identified by using the LBID which is evenly distributed and the workload of each ID space is balanced. This LBID is dynamically assigned and self-organizable without any manager and the LBID routing table which helps for routing to any representative nodes is also organized when the LBID is assigned. The transient nodes join/leave very frequently on the system and the majority of the participant nodes are these transient nodes. These nodes act as simple roles such as servicing the request and helping the representative nodes. For this purpose, the transient nodes are identified by the LFID which makes the ID region of a transient node as small as possible and reduces the effect of the dynamic membership change. The frequently joining/leaving of transient nodes hardly affects the p2p system. Because the majority of churn is the churn of transient nodes, the behavior-aware p2p protocol can reduce the maintenance overhead of the whole p2p system and achieve more efficient routing without the frequent updates.

Moreover, when a new nodeID is assigned, the LBID table and LFID table are also organized by self-organizing manner. To find the location of any ID(nodeID or objectID), the LBID table is used for routing to the correct sub-region which is mainly guaranteed by a representative node and the LFID table identifies the right node. The LBID table is composed of $O(logN_r)$ entries where N_r is the number of representative nodes. This is one of the distributed hash table and its lookup performance is also similar to the DHT. The LFID table obtains the node information on the same sub-region and it needs just o(1) cost for the lookup. According to this, the behavior-aware p2p provide the efficient and bounded lookup as same as the DHT based p2p.

This dissertation is organized as follow. In chapter 2, the DHT based p2p algorithm and the other researches which try to reduce the maintenance overhead are described. Chapter 3 shows the naive replication approaches to reduce the maintenance overhead. These approaches modify the consistent set of DHT based p2p which is mainly used to replication. Chapter 4 introduces the detail of the behavior-aware p2p protocol. The key part is the assignment of the dynamic nodeID. During this assignment, each node gets its proper nodeID according to its behavior on the p2p system. This protocol supports the efficient data replication management and the correct and efficient lookup. The simulation environment and performance evaluation are given in chapter 5. The behavior-aware p2p can save about 80% data traffic in normal case and keep the high data availability. The efficient lookup is also provided. Chapter 6 provides concluding remarks.

2. Background

2.1 DHT based P2P Protocols

There are many DHT based p2p algorithms such as Chord, Pastry, Tapestry and CAN [5][6][7][8]. Each node has a DHT which is a small routing table and any node can be reached in about $O(logN)$ routing hops where the N is the total number of nodes in the system. To achieve this efficient and bounded routing, there are some rules for the organizing the participant nodes. First of all, each node has a unique nodeID which is taken by hashing any identifier of a node, and according to its nodeID it maps on the ID space where the nodes and the objects are co-located with the nodeIDs or the keys which are the hashed values of the nodes or the objects. In the figure 2.1, the nodeID of node A is 3 and it maps on the position for 3. After the mapping of the node id, each node knows its ID region from the next position of its previous nodeID to its nodeID and each node should store and service the objects for its ID region. In the figure 2.1, node A takes its ID region from 1 to 3 because its nodeID is 3 and its previous node B locates on 0 and when a node wants to get a.mp3 whose key is 2, node A gets the request for it.

The Chord[5] project uses a logarithmic-sized routing table to route object queries. Its main idea is constructing ring-like network. For a namespace defined as a sequence of m bits, a node keeps at most m pointers to nodes, which follow it in the namespace by 2^1, 2^2, and so on, up to 2^{m-1}. The i_{th} entry in node n's routing table contains the first node that succeeds n by at least 2^{i-1} in the namespace. Each data is stored on the first node whose identifier is equal to or immediately follows its key in the namespace.

In Pastry[6], nodes are responsible for keys that are closest numerically.

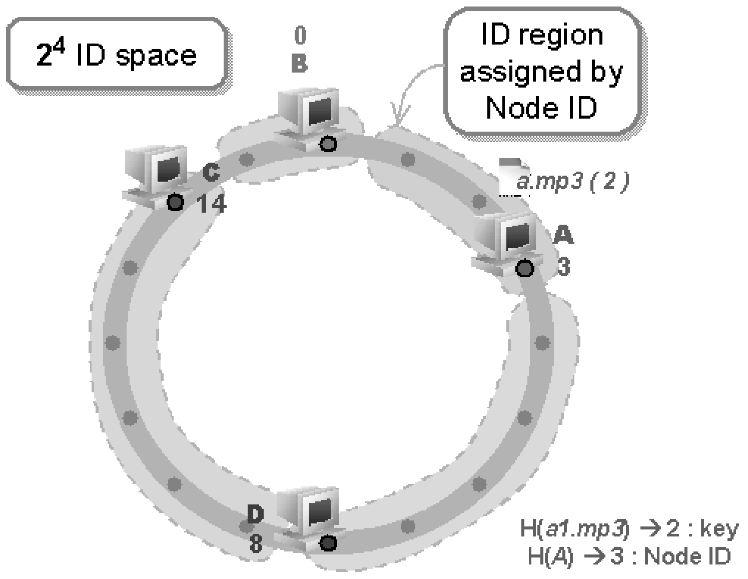


Figure 2.1: Overview for the general DHT based P2P algorithm

When presented with a key, a node routes the query to the node with a nodeId that is numerically closest to the key, among all currently live nodes. Each node keeps track of its immediate neighbors in the nodeId space, called a Leaf Set L which is the set of $|L|$ closest nodes (half larger, half smaller). This set ensures that the routing can be achieved correctly. Because of that, in spite of any failures, each node maintains the correct Leaf Set.

Pastry node has another set of neighbors, called a Routing Table, whose entities spread out in the key space. A Routing Table is organized into $(log2bN)$ rows with $2b-1$ cols each (N means the maximum nodeId). The $2b-1$ cols in row n each refer to a node whose nodeId matches the present node's nodeId in the first n digits, but whose $n+1_{th}$ digit has one of the $2b-1$ possible values other than the $n+1th$ digit in the present node's nodeId. By using these neighbors, I can achieve the routing more efficiently. Routing consists of forwarding the query to the neighboring node that has the longest shared prefix with the key. And, in the case of ties, a key is forwarded to the node with identifier closest numerically to the key. Consequently, pastry has $(log2bN) * (2b - 1) + 2L$ neighbors and routes within $O(log2bN)$ hops.

Tapestry[7] architecture is suggested to enhance Plaxton to be used as a routing architecture in peer-to-peer environment. In Tapestry's algorithm, every node is assigned a unique ID that has n bits. Tapestry guarantees that the maximum hop count to route message is the logarithm of the node ID space, and the average value is the logarithm of the number of nodes with logarithmic overhead of table maintenance. Hence, it is scalable to the increase of node count.

It is also self-organized without global knowledge about the entire network. Despite the absence of global information, routing algorithm of Tapestry guarantees that routed destination from the different places with the same destination ID is always the same node. Tapestry is more robust than Plaxton as it maintains more neighbors in the table and periodically checks the health of the neighbor nodes.

In the "Content Addressable Networks" (CAN)[8], nodes are mapped onto a d-dimensional coordinate space. This coordinate space is completely logical. The space is divided among all the nodes in the system, so every node owns individual space. Each node keeps (key, value) pairs and information of its neighbors. Key is deterministically mapped onto a point in the coordinate space using a uniform hash function. Then, node that includes the point stores this (key, value) pair. If a node wants to retrieve a value using a key, it determines the point that includes this key using the same uniform hash function, then sends lookup request message to its neighbor towards the destination coordinate. Because every node keeps information of its neighbors, lookup message request message is routed to destination node.

CAN routes messages in a d-dimensional space, where each node maintains a routing table with $O(d)$ entries and any node can be reached in $O(dN1/d)$ routing hops. In contrast to other peer-to-peer lookup system (Chord, Pastry, Tapestry, and etc.), the state maintained by a CAN node does not depend on the network size N, but lookup cost increases faster than the others ($logN$). However, if $d = logN$, lookup cost is similar.

Freedman *et al.* propose a new lookup mechanism based on a distributed trie. In the scheme, nodes dynamically replicate only partitions that frequently access while in previous DHT-based mechanisms each node statically assigns only those partitions that are close to its address. In addition, they used lazy updates of lookup structure to reduce maintenance cost by piggybacking trie state. But they cannot limit the degree of inconsistency and the number of lookup hops may be very large without upper bound.

2.2 Node Heterogenity

The peer-to-peer systems are composed of components with extremely heterogeneous availabilities - individually administered host PC's may be turned on and off, join and leave the system, have intermittent connectivity, and are constructed from low-cost low reliability components. For example, one recent study[11] of a popular peer-to-peer file sharing system fond that the majority of peers had application-level availability rates of under 20 percent and only 20 percent nodes have server-like profiles. In such an environment, failure is no longer an exceptional event, but is a pervasive condition. At any point in time the majority of hosts in the system are unavailable and those hosts that are available may soon stop servicing requests. In this case, when a node joins/leaves system, the affected data structure on some existing nodes must be updated accordingly to reflect the change. For nodes which join/leave the system frequently, system will generate a lot of routing information update traffic and data copy traffic. It is not only increase the bandwidth consumption, but also affect the efficiency of DHT based routing algorithms.

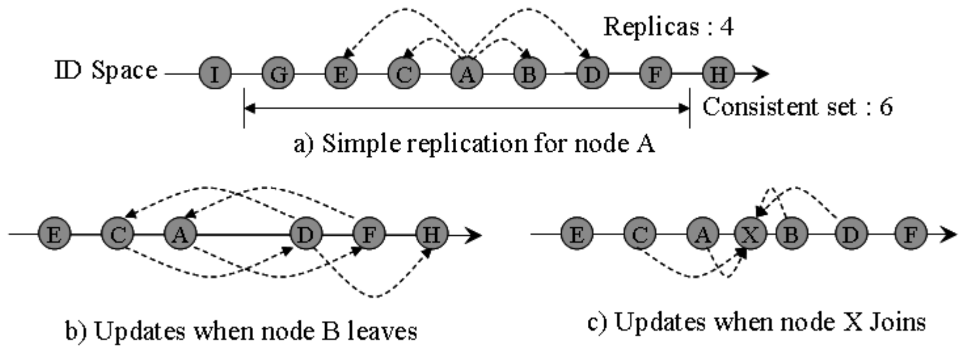

Replicas : 4
ID Space
I
G
E
C
A
B
D
F
H
Consistent set : 6
a) Simple replication for node A
b) Updates when node B leaves
X
c) Updates when node X Joins

Figure 2.2: Simple replication in DHT based p2p

2.3 Data Availability

These DHT based algorithms can lookup any data efficiently with DHT, but when the massively node failures occurs and spoils the information of DHT without any notification, this efficient lookup can not guarantee the correctness. To cope with the massively node failures, they use the consistent set such as the successor list of chord and the leaf set of pastry. This consistent set of a node is composed of the neighbor nodes which locate numerically near to the node on ID space. This set is tightly coupled to the current state of nodes and when any node joins or leaves, the consistent set of every node which detects the change of the membership must update to preserve the current state of the p2p system. The p2p system guarantees the correctness unless all members of consistent set fail simultaneously.

This consistent set is used for not only the routing correctness but also the data availability on durable p2p storage systems such as p2p file systems [9] [10] and p2p file sharing systems [1]. Data Availability means the total availability when the multiple nodes have the data. This availability is obtained by subtracting the probability of that all nodes which have the data leave from 1. That means if only one node is alive, the data is available. Like the figure

2.2(a), one node replicates stored objects to the neighbor nodes which are the member of consistent set until the replicas are enough to achieve target data availability. This simple replication guarantees the simple data availability management and the simple lookup under churn easily and automatically. Because the consistent set has the current state of nodes and updates immediately under churn, the p2p system keeps the target data availability automatically. Moreover, because neighbor nodes of a node already have the replicas of its objects, even if this node leaves, the neighbor node automatically replaces it as a servicing node for its object range without additional object copies.

However, the simple replication causes the more maintenance traffic under churn. If the number of replicas is N and a node leaves, the new $N+1$ replicas are needed for the affected nodes. In figure 2.2(b), when a node B leaves, the nodes A, C, D, F which already have the replicas on node B should make new replicas and node D which is newly responsible for the object range of node B makes the replica for this range additionally. Moreover, when a node joins, each affected node copies the objects to it as a new replica like figure 2.2(c). In this case, when node B joins and leaves very frequently, the compulsory data replications occurs and the heavy data traffic wastes even if the dynamic behavior of node B can not affect the data availability. According to this simple behavior and the heavy churn of the p2p participants, the data traffic needed to support the simple replication is very high and bursty.

The commercial p2p file sharing systems leave the data replication up to the popularity of the data. The popular data is replicated on many clients and the data availability of this data is very high. However, the unpopular data are stored on few clients and it is very hard to find this data because of very low data availability. To make the p2p storage system durable, the smart data replication methods is needed and the each inserted data is available for any time and has the similar data availability.

In the paper[12], they stores the replicas on the random nodes on the ID space and periodically checks their availability. This behavior reduces the compulsory copy because the replication has no relation to the consistent set. However, this approach takes too much control traffic to keep the node availability of all replicas for every object on the system. Moreover, they do not use the consistent set and the change of the data availability caused by the node failure is detected slowly. The paper [13] shows that the erasure coding approach reduces the traffic of the replication by using the computing power.

2.4 Data Maintenance Overhead

Though these well-organized rules make the routing of the p2p system efficient and bounded, a big issue in current DHT based p2p systems is the high information maintenance overhead of maintaining DHT routing data structure and the stored data. Because its nodeID is already given by the hashing function and its position on ID space is already fixed, when a node joins/leaves the p2p system, the ID region of its neighbor nodes changes and the stored data should be copies for the new ID region to service the right and reliable object, and the update of the routing table is also needed. In figure 2.1, if node A leaves, the ID region of node D changes and the object from 1 to 3 should be copied from node A to node D. Moreover, the affected routing table which has the entry with node A must be updated. In this case, one recent research[11] of a popular p2p file sharing system found that 80 percent of total nodes of a p2p system join/leave very frequently and the majority of nodes have the application-level availability rate of under 20 percent. In such an environment, the information maintenance overhead is getting worse and this overhead discourages that the DHT based p2p systems are deployed to the real world.

Some researches emerged to prevent the information maintenance overhead by using the heterogeneity of participant nodes. In the paper [14], they manage the DHT which is certain amount of system routing information with the availability of each node which is evaluated during the time it joins the system. They proffer to add stable nodes into routing data structures instead of frequently join/leave nodes. The paper [12] tries to reduce the compulsory data copies for joining/leaving nodes with the node availability. They manage the availability of each data by evaluating the availability of each node which stores the data. The common feature of these approaches is that the stable nodes take most system workload and this reduces the information maintenance overhead of the DHT based p2p systems. However, in these approaches, though the stable nodes get too much workload, they lack for the explicit method which balances the workload of each stable node. Because each stable node already has the fixed nodeID and the space between any two stable nodes is unbalanced, each node gets the unfair workload. Moreover, the fixed nodeID still affects the ID region of a node and the joining/leaving for a node makes the compulsory copies too. In my solution, the behavior-aware p2p protocol which uses the dynamic nodeID, each reliable node gets the balanced ID region and workload and normal nodes which join/leave very frequently affects the information maintenance overhead little.

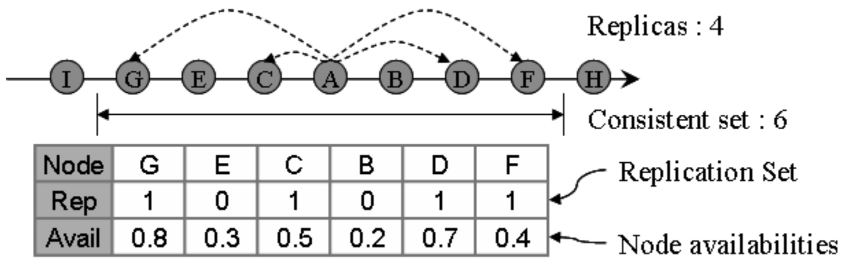
Some approaches [17][18] provide the hierarchical method to use the reliable node as the supernode. However, because these methods assume that the powerful supernodes already exist, they lack for the explicit method to sort out the reliable nodes from the whole participant nodes. This make the system inflexible and the problems of the static nodeID also exist. The new behavior-aware p2p protocol assigns the nodeID to a node dynamically with time and each node can change its nodeID easily on the fly. This behavior makes that the system chooses the reliable nodes and each node moves to the proper position according to its characteristics easily and automatica.lly

3. Naive Replication Approach

3.1 Quorum based Replication

The simple replication method basically uses the concept of the quorum. The quorum means that the fixed minimum number of members of a set which must be present for its objective to be valid. That is, if the number of the replicas for an object is more than the target quorum, the p2p system considers that the object is durable under massive failures. However, this simple method directly uses the consistent set which is tightly coupled to the current state of networks and when the membership of nodes changes, each affected node should update its replicas and performs compulsory copies to preserve data durability. These compulsory copies take high and bursty traffic under churn. Sometimes, this copies for replicas are meaningless because the new replicas leave soon.

To prevent these compulsory copies, I propose the quorum based replication method which is loosely coupled to the consistent set. Like the figure 3.1, the new information is added; the replication set that indicates which node replicates the object. The range of this set is same to the consistent set, but the update of this set occurs individually. When a node joins/leaves, the affected consistent set should be updated for the correctness of the routing mechanism, but the replication set is only updated when the number of replicas is fewer than the target quorum. Because the size of the consistent set is generally bigger than the number of replicas, the replicas can be interleaved on the consistent set like the figure 3.1 and they are loosely coupled to the current state of networks. If the number of replicas is N and a node leaves, the number of affected nodes can be less than $N + 1$ and the compulsory copies can be



Node	G	E	C	B	D	F
Rep	1	0	1	0	1	1
Avail	0.8	0.3	0.5	0.2	0.7	0.4

Figure 3.1: Metadata for simple replication methods

reduced under churn.

When a node needs a new replica, it selects the numerically closest node from this node as a replica, because a member node which is numerically farther from this node withdraws from the consistent set with higher probability under churn. In the figure 3.1, if node A needs one more replica, node B can be selected as a new replica rather than node E.

3.2 Availability based Replication

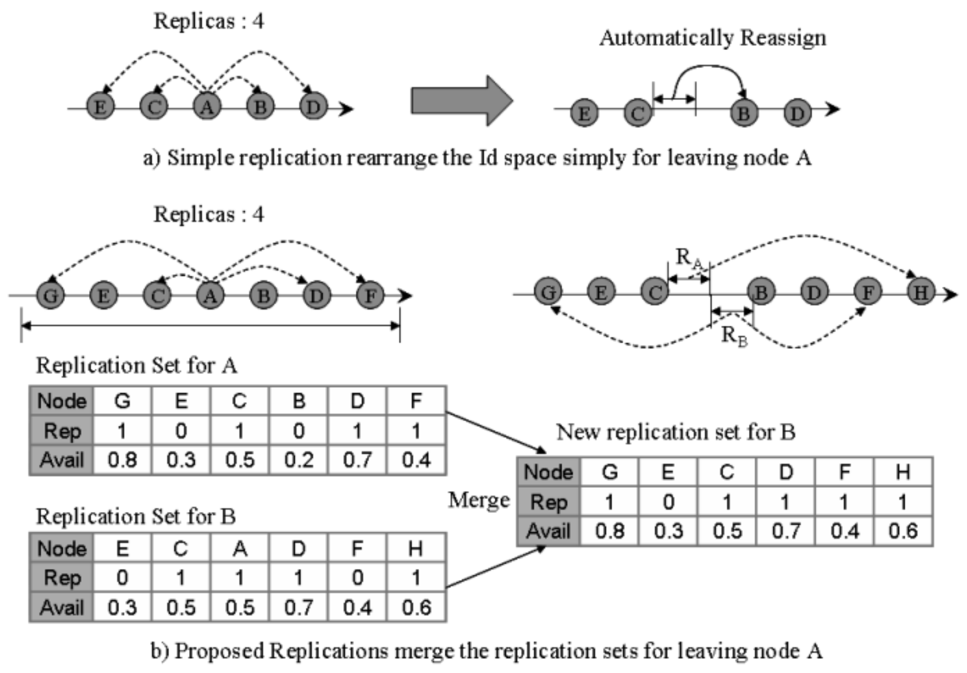
The quorum based replication tries to keep the number of replicas above the target quorum to achieve the target data durability. However, if a new replica is assigned on a node which has the low availability, this node may leave soon and another new replica is needed. If a new replica is chosen with the node availability, the more available node is selected as a replica and can reduce the overhead. I assume that the node availability is the prediction value how long a node is alive after it joins the p2p system, because in other researches[11] the long lived nodes generally have the large bandwidth and the big computing power. This availability information is computed by each node and each node advertises this information to all members of the consistent set by using the piggyback method. To detect the node failure, a node periodically sends a ping message to all member of the consistent set. The availability information is piggybacked on this ping message and each node manages the node availability of the member of the consistent set with little cost.

Like the quorum based replication, the availability based replication uses the replication set to make that the replicas are loosely coupled to the consistent set. The main difference of these replications is the selection mechanism for a new replica. In the availability based replication, each node computes the data durability with the node availabilities of replicas and the replication only occurs when the data durability is below the target threshold. When a new replica is needed, a node selects the most available node of the consistent set.

3.3 Management of Replication Set

In the figure 3.2(a), when node A leaves, the lookup request for the object in node A is forwarded to node B automatically because of the basic concept of the DHT based p2p. In the simple replication, node B always has the replicas for node A and already has the objects of the responsible range of node A (R_A). That is, the rearrange of the ID space performs easily and automatically. However, in the quorum based and availability based replications, the replicas are interleaved on the consistent set and it can not be sure that node B has the replicas for node A. In order to copy the objects of the new responsible range, node B should know the locations of the replicas for node A. To do this, each node advertises its replication sets to all members of its consistent set with piggybacked ping message.

Under churn, the change of the object range affects the replication set. The figure 3.2(b) shows the management of the replication set when node A leaves. The closest neighbor node B which is automatically forwarded the message for node A will be responsible for the object range of leaving node A (R_A). At this point, there are two replication sets for node A and node B and these sets


Replicas : 4
E C A B D
Automatically Reassign
E C B D
a) Simple replication rearrange the Id space simply for leaving node A
Replicas : 4
G E C A B D F
G E C B D F H
R_A
R_B
Replication Set for A
Replication Set for B
Merge
New replication set for B
b) Proposed Replications merge the replication sets for leaving node A

Replication Set for A

Node	G	E	C	B	D	F
Rep	1	0	1	0	1	1
Avail	0.8	0.3	0.5	0.2	0.7	0.4

Replication Set for B

Node	E	C	A	D	F	H
Rep	0	1	1	1	0	1
Avail	0.3	0.5	0.5	0.7	0.4	0.6

New replication set for B

Node	G	E	C	D	F	H
Rep	1	0	1	1	1	1
Avail	0.8	0.3	0.5	0.7	0.4	0.6

Figure 3.2: Management for the replication set when node A leaves

should be merged to one new replication set for node B. The basic rule of the merging operation is that if a node is a replica for any set, it remains a replica for the new set. In the figure, node F is a replica for node A and it becomes a new replica for the new replication set. Otherwise, node E is not a replica for any set and it is still not used as a new replica too. After merging operation, the new replication set can have more replicas than each previous replication set and the data durability increases automatically.

During this merge operation, the real objects are selectively copied to the replicas of the new replication set. Node H which is the previous replica of node B already has the object for R_B and the objects for R_A are only copied to node H when the merging occurs. By the same reason, node F and node G which are the previous replicas of node A only obtain the objects for R_B during the merging operation. This selective copy can reduce the traffic overhead of the replication.

When a new node joins and a target node gets this join request, the object range of the target node is divided into two object range and the new node is responsible for one of them. In this case, the new node simply copies the replication set of the target node and adds the target node as a new replica because it already has the object for this range. The target node also adds the new node as a new replica.

3.4 Limitation of Naive Approaches

These approaches use the consistent set which is the basic information of DHT based p2p. They only modify the usage of the consistent set and reduce the information maintenance overhead to keep the high data availability. That is, according to these approaches, the replicas are loosely coupled to the consistent set and they are interleaved on the consistent set. When churn occurs, some nodes can not affect the data availability and the p2p system can reduce the maintenance traffic.

However, these approaches has a critical hurdle to reduce more maintenance traffic. That is the static nodeID of DHT based p2p and the consistent set which is used to replicate the data. Every node gets its nodeID statically by hashing the identifier of a node such as ip address and mac address. They locate on the ID space accordin to its nodeID. Moreover, the range of the replication is bounded by the range of the consistent set. In this case, if a node has member nodes which have high node availabilities for the consistent set, this node can save the data traffic with this naive replication approaches. However, if a node has the consistent set in which the member nodes only have low node availabilities, these member nodes join/leave very frequently and this node which uses the naive replication approaches can not reduce the maintenance traffic effectively. In DHT based p2p can not guarantee that the high available nodes are distributed well-balanced. Consequently, only the modification of the consistent set can not guarantee that these naive approaches reduce the maintenance data traffic. To solve this problem, high available nodes should be distributed well and each node should be located the proper position according to its behavior.

4. Behavior-aware P2P Protocol

4.1 Overview

Previous DHT based p2p systems lack the explicit methods for exploiting the heterogeneous characteristics of participant nodes. The main reason of this lack is the static ID which makes the location of a node fixed on the ID space, and the system with the static ID is not flexible. On the left side of figure 4.1, the large sized computer means the reliable and powerful node and the small sized computer presents the normal and weak node which frequently joins/leaves. Because each reliable node already has the fixed nodeID and the space between any two reliable nodes is unbalanced, each node get the unfair workload. Moreover, the fixed nodeID still affects the ID region of a node and the joining/leaving for a node makes the compulsory copies too.

I address this problem with the dynamic nodeID which changes according to the characteristics of a node with time. The participant nodes are classified into two types : representative nodes and transient nodes. The representative nodes are more reliable and more powerful nodes and the transient nodes join/leave very frequently. The nodeID of a representative node is well distributed on the ID space and makes that the each representative node gets fair ID region and balanced loads. The transient node gets the nodeID which makes its ID region as small as possible to minimize the information maintenance overhead for joining/leaving of it. According to this behavior-aware approach, the representative nodes are sorted out automatically and these nodes get well balanced loads like the right side of figure 4.1. The transient nodes also are picked out and the churn of them hardly affects the maintenance overhead to keep the high data availability.

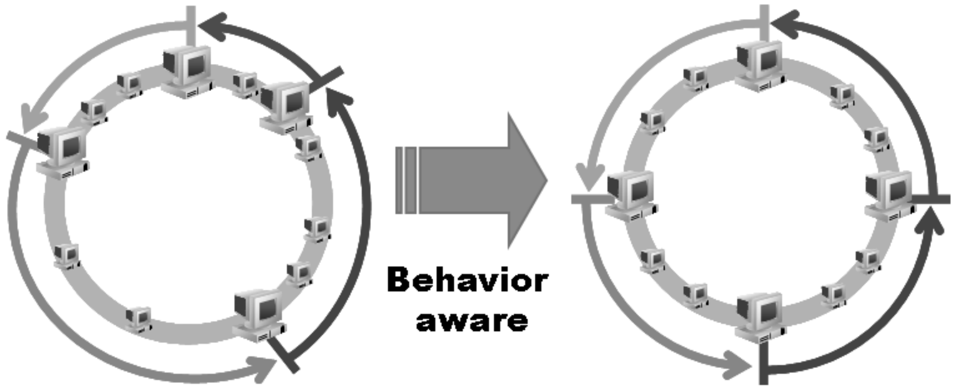
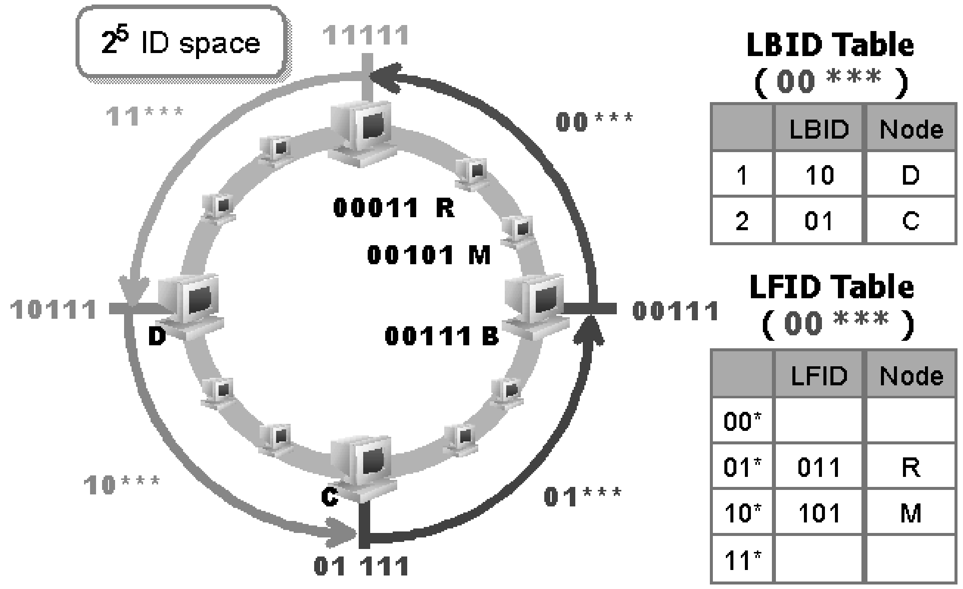


Figure 4.1: DHT based P2P vs Behavior-aware P2P

Figure 4.2 shows the stable state of the p2p system that uses the dynamic nodeID based p2p system. In this figure, the large computer means the representative node and the small computer means the transient node. The participant nodes are on the 2^5 ID space and the number of bits for a nodeID is 5. The general systems use 2^{128} ID space, but in this example only few bits are used for the easy explanation. To distribute the participant nodes efficiently, the ID space should be divided into many *sub-regions* which are the balanced ID regions. Each sub-region has one representative node which represents this region and many transient nodes which assist the representative node. That is, the representative node is mainly responsible for routing and servicing the objects for the sub-region and the transient nodes service the objects for the small ID region which is assigned by their nodeIDs.

The nodeID consists of the *Load-Balanced ID (LBID)* and the *Load-Free ID (LFID)* and ,in this example, the first 2 bits of a nodeID means the LBID and the other 3 bits is for the LFID. The LBID is the identifier for the sub-region and the LFID is the identifier for the node. All nodes on the same sub-region have the same LBID and they are identified by the LFID. The all bits of LFID for the representative node are set to 1 and the LFIDs of other



LBID Table
(00 *)**

	LBID	Node
1	10	D
2	01	C

LFID Table
(00 *)**

	LFID	Node
00*		
01*	011	R
10*	101	M
11*		

Figure 4.2: Overview of Behavior-aware P2P Protocol and LBID/LFID tables for node B(00111)

transient nodes change according to the behaviors of them. In the figure 4.2, nodes B, R and M are included in the same sub-region and their LBIDs are all same value, 00. However, the representative node B has 111 LFID and the other transient nodes gets different LFIDs.

During the initial state of behavior-aware p2p, new nodes are assigned as representative nodes to be responsible the sub-region. In this case, the static ID can not be used at all and every nodeID are assigned by the dynamical manner to distribute the load of each sub-region evenly. After this state, when a node joins, it uses its static nodeID by hashing its identifier like any previous DHT algorithm. This hashed values of nodes is only used to find the sub-region for them and to distribute evenly them to the sub-regions. To route to the right sub-region, a node which gets a join request forwards it to the next node which is the node of the most prefix matched entry of the LBID table. After finding the sub-region, the LFID table assigns the right LFID to the new node. In the next section, the detail of the whole join process is described.

If the node B leaves, one node which is considered most reliable and available among other nodes on the same sub-region replaces the node B as a new representative node and it changes it nodeID for the new position. According to this behavior, a node which joins or leaves frequently is hard to be a representative node and even if it locates on the position of a representative node fortunately, it leaves the system easily and the more reliable node replaces it.

When a node tries to lookup an object, it sends a lookup request with the object key to any other participant node. Like the case of the join process, it forwards to the right sub-region by the LBID tables and find the node whose ID region is responsible for the key by the LFID tables. For example, in figure 4.2, when a node wants to get an object whose key is 00001, the representative node B takes the request, However, when a node tries to get an object whose key is 00010, the transient node R takes the request to assist the representative node, because the ID region of node R is from the start of the LFID slot,00010 to its nodeID, 00011.

4.2 Dynamic NodeID

4.2.1 NodeID Transformation

In the behavior-aware p2p, the nodeID is not static but dynamic. When a node joins, it gets its nodeID to locate on the ID space, however, it change its nodeID with time to organize the efficient p2p system according to its characteristics. The figure 4.3 shows this ID transformation. At the first time, when there are few nodes in the p2p system and any one of sub-regions needs a representative node, this phase is called the bootstrap phase and in this phase the LBID assignment performs to distribute the sub-region evenly and to make up the LBID routing tables. When a new node joins, regardless of its

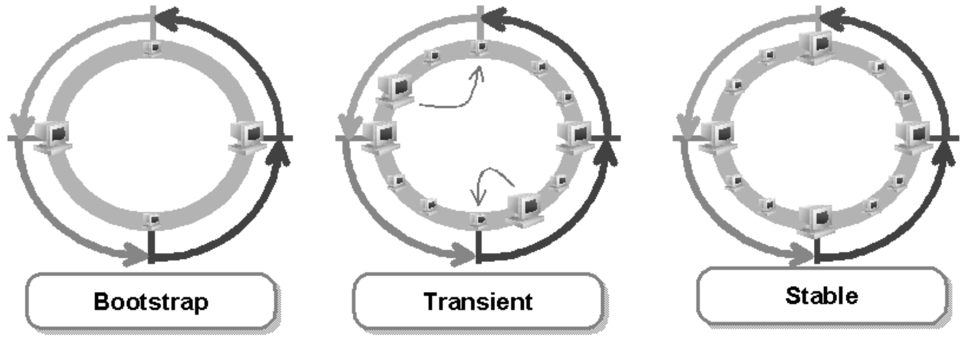


Figure 4.3: Three phases of nodeID assignment

behavior and characteristic it should get nodeID to represent the sub-region as representative nodes. According to this, in this phase some representative nodes are not a really reliable and available node.

After the bootstrap phase, that is, when the every sub-region has the representative node, the transient phase starts. In this phase, some representative nodes which is not reliable are displace and new reliable nodes which are in the same sub-regions replace them. To do this, new nodes get nodeIDs to assist sub-regions as transient nodes by performing the LFID assignment which assigns little load to transient nodes. When a new node joins, the join message is forwarded to a target node which locates on the right sub-region for the new node. The target node assigns the new LFID to the new node and fills the LFID table. When the representative node fails, the most reliable transient nodes in the same sub-region substitute it.

According to this assignment and the failure recovery, the p2p system becomes stable. That is, in the stable phase, more reliable nodes acts as representative nodes and the other transient nodes assist the representative nodes as transient nodes. Like the transient phase, in the stable phase, new nodeID is assigned by the LFID assignment and the same failure recovery is used.

4.2.2 NodeID Assignment - LBID

During the LBID assignment, the behavior-aware p2p protocol can not use the static nodeID which is generally used by other DHT p2p algorithm and should assign the proper nodeID to each representative node without a help of any servers which manages the proper nodeIDs. This LBID is assigned after a new node finds any one of reliable nodes. The LBID of the new node are assigned by the reliable node which gets a join message and the LFID bits are set to 1 because it acts as a representative node. In this LBID assignment, the new node also creates the LBID routing table according to its LBID. Each reliable node has the state information such as Join, Level, Full and Leaf. When the Join bit sets to 1, this node can process the join request and create new LBID for the new node. The Level bit is the depth value which means how many join requests is processed in this node, that is, how many routing entries are filled. The Full bit sets to 1 after the enough reliable nodes join the p2p system and they are ready to get the transient nodes. The Leaf bit means the number of transient nodes which is connected to a reliable node. According to these state information, LBID is assigned automatically and correctly.

The basic algorithm for the LBID assignment is in figure 4.4. When a node joins to the system and there is no reliable node, the new node has the new LBID whose all bits set to 1. Otherwise, when any reliable node gets a join request, it creates new LBID based on the two information which are its LBID and the Level bit. That is, the $level_{th}$ bit of LBID sets to the exclusive bit and this is the new LBID. This simple rule makes the difference of LBID of any two closest reliable nodes even and each reliable node gets the balanced and fair ID region. The LBID routing table which is used for routing to any reliable nodes is also organized when the new LBID is created. The basic rule is the N_{th} entry of the routing table has the node information whose N_{th} bit of LBID is exclusive to the owner's LBID. These bit-wise exclusive entries make the LBID routing table and any node can reach any other nodes. In figure

```
If No Representative Node
    $LBID_{new}$ = set all bits of LBID to 1
    Level++
Else
    If( !Full ){
        // Bootstrap phase
        If( Temporal Routing Entry )
             Forward Join request to this temporal entry
        If( Join ){
            // Accept Join
            $LBID_{new}$ = set exclusive bit of $Level_{th}$ bit of $LBID_{target}$
            $Level_{th}$ RT entry = $LBID_{new}$
            Level++
            If( Level > Level_Thres) Join = 1
        }
        Else{
            If( sending node != last RT entry )
                Forward Join request to next RT entry of sending node
            Else
                If( Find_Joinable_Representative_Node() )
                    Forward Join request to this representative node
                else
                    // finalize
                    Sending Notification of Finalization (Full = 1) to all nodes
        }
    }
    Else{
        // Transient & stable phase
        Find most prefix matched LBID RT entry with the static NodeID
        If( LBID match )
            LFID assignment with LFID Table
        Else
            Forward Join Request to the RT entry
    }
```

Figure 4.4: Basic algorithm of the NodeID assignment

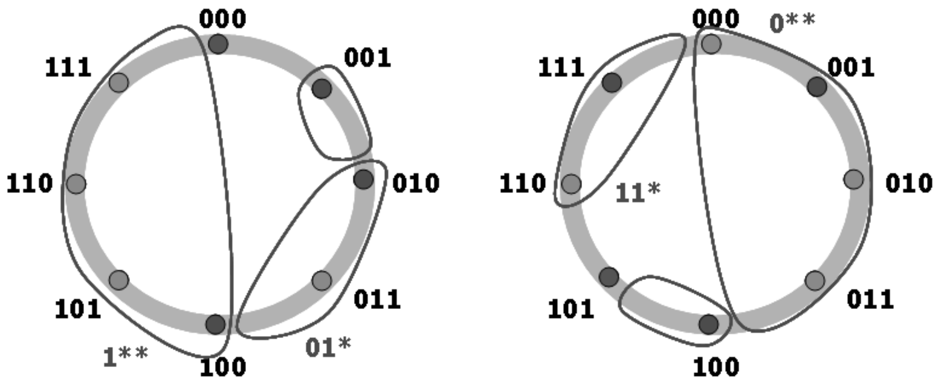


Figure 4.5: The basic concept of LBID routing table (left: LBID 000, right: LBID 101

4.5, the node whose LBID is 000 has routing entries 100, 010, 001. In this case, the routing entry 100 is the next routing point for LBID 1** and the routing entry 010 is the next routing point for LBID 01*. According to these routing entries, each node can reach any other nodes by LBID. Like figure 4.5, this LBID routing table has $logN$ of routing entries and the maximum routing hops are limited to $logN$, where N is the number of LBID bits. When there is not proper node information for a routing entry, this entry becomes the temporal routing entry. In figure 4.6, the first entry of node B is C(101) which is the right one. However the second entry of node B is G(110) which does not fit to this entry and this entry is called the temporal routing entry. This temporal routing entry has the node information which does not matched but closest to the right node information. When the node which has the temporal routing entry gets a join request, it forwards the join request to the temporal node which is ready to process join request. After this forwarding process, the new node replaces the temporal routing entry with right routing entry and the LBID routing table is composed completely.

The figure 4.6 shows a simple example of the LBID assignment. When the new node A joins and the target node B gets this join request, the node B makes a decision which it makes a new nodeID or forward to next nodes. The main point of decision is the LBID routing table and the target node tries to fill in the LBID routing table. In this case, the node B treats this join request, because its third routing entry is empty. According to the basic rule for routing table which is described on the previous section, the target node B assigns new nodeID, 000 to the new node A to fill the third routing entry. After the new node A gets new nodeID, it also makes up the LBID routing table. To do this, when the target node B responses the request, it gives not only the nodeID but also its new modified LBID routing table. The node A checks that each delivered routing entry is the right information for its LBID routing table according to the basic rule for routing table. If the routing entry is the right one, it is used, otherwise, it tries to find the right information by requesting to the wrong one. For example, the third entry of the node A is B(001) which is the delivered information from B because of it is the right one. However the second entry is D(010) which is the first routing entry of the node G, because the node G is not fit for the second entry of the node A and the node A requests the proper information to the node G. Moreover, the first entry of the node A is the temporal routing entry. If one node tries to find the proper node information but it can not, it fill the entry with the temporal information. This temporal information means that this node may treat the join process because this temporal information should be fixed with the right one. After the new node updates its LBID routing table, it should advertise its information to every node of its LBID routing table to fix the temporal information.

I describe the join process when the LBID routing table of target node is not full. If the LBID routing table full, it should forward the request to the next node which can deal with the join request. First of all, the target node looks into any temporal routing entries and if there is anyone, the target

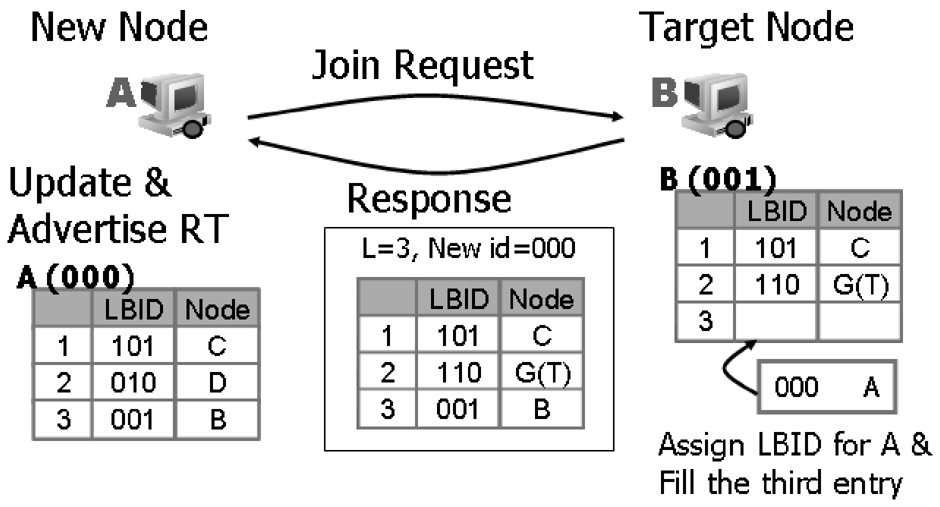


Figure 4.6: Simple example of the LBID assignment

node forwards the join request to the node which is referred by this temporal routing entry. Otherwise, if there is no temporal routing entry, the join request is forwarded by referring to the LBID routing table with descending manner. At first, a node whose routing table is full forwards the join request to the first routing entry. If the next node which gets the join request also has the full routing table, it sends the join request to the second routing entry because the first entry has the information of the sending node. According to this descending routing mechanism, the join request is forward to a joinable representative node.

If the join request is forwarded by referring the last routing entry of any node, this node tries to find the target node which can treat the join request among the whole of the representative nodes. In this case, each node just does not know the whole of the representative nodes, but only knows the L routing entries where L is the number of the LBID bits. According to this, one nodes can not notify to all other nodes by itself and the p2p system needs the efficient and systematic method to visit the whole of the representative

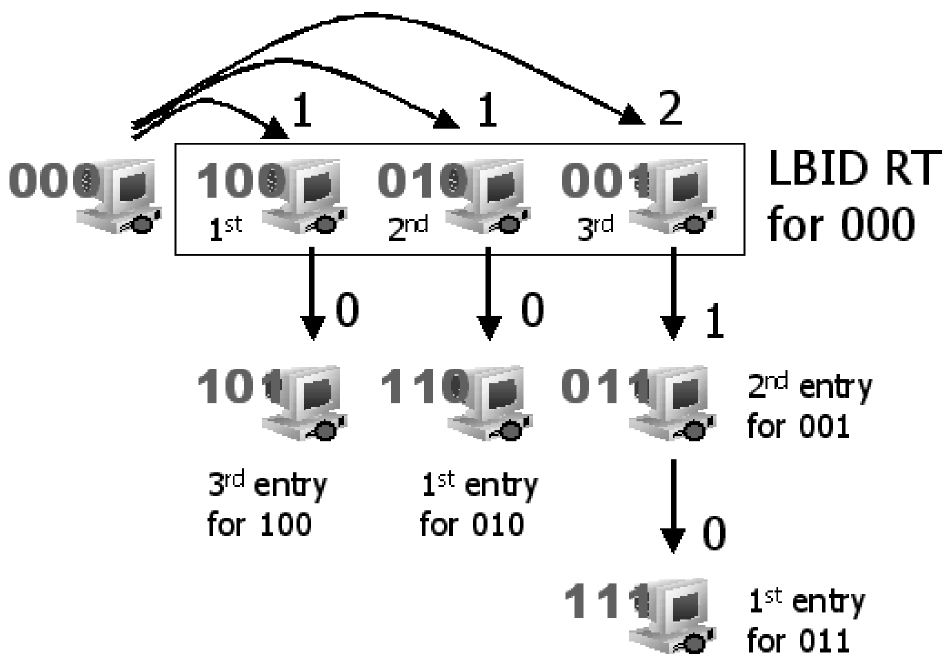
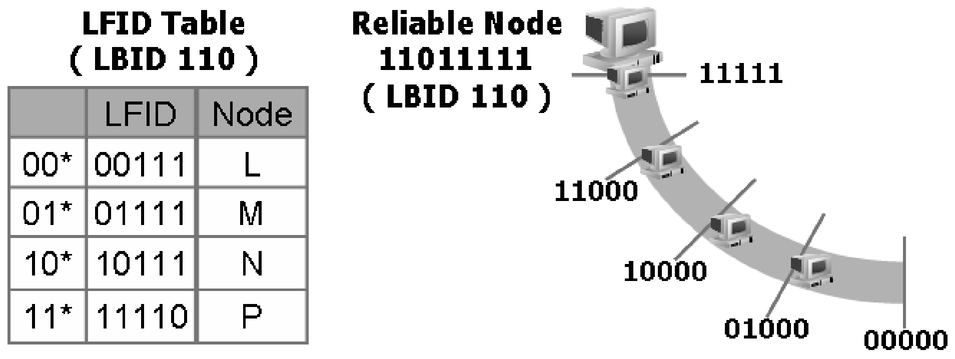


Figure 4.7: Message flow to the whole of the representative nodes

nodes. To achieve this, a node sends messages with TTL count value to every nodes of the routing table. Like figure 4.7, the message for the N_{th} routing entry has $N-1$ TTL count value, except the 1st entry for whom the message has 1 TTL value. The target node which gets the message reduces the TTL value by 1 and the message is forwarded to the $M-1_{th}$ routing entry, where M is the position of the target node on the routing table of the sending node. For example, if the target node is 2nd routing entry of the sending node, this target node sends the message to its first routing entry. However, if the target node is 1st routing entry of the sending node, it sends the last message to its last routing entry. According to this behavior, one node can visit the whole of the representative nodes. If there is no target node which can process the join request, the LBID assignment finish and the finalize mechanism starts. This finalize mechanism visits all of the representative nodes and set the Full bit to



	LFID	Node
00*	00111	L
01*	01111	M
10*	10111	N
11*	11110	P

Figure 4.8: LFID Table and node assignment (LBID 110)

1 for every representative node to be ready to get the transient nodes.

4.2.3 NodeID Assignment - LFID

After the bootstrap phase, the transient phase starts. In this phase, each node joins in the system as the transient nodes. Because the transient nodes are generally composed of the nodes which join/leave frequently, the p2p protocol should minimize the responsibility of them to reduce the compulsory management cost such as the updates of the routing tables and the data copy for the responsible data and the replicated data which are due to the dynamic membership change.

Basically the transient node acts as an assistant for the representative node. Each transient node locates on a subregion according to its static nodeID which is generated by SHA hashing. Each subregion has LFID table to identify the region of the transient node and each transient node fill this table. In figure 4.8, the subregion whose LBID is 110 has a LFID table like that. A LFID table has 4 slots which is identified by the prefix such as 00*, 01*, 10* and 11*. When a transient node join, the empty slot is assigned to this transient node. Each transient node is responsible for storing and requesting the data

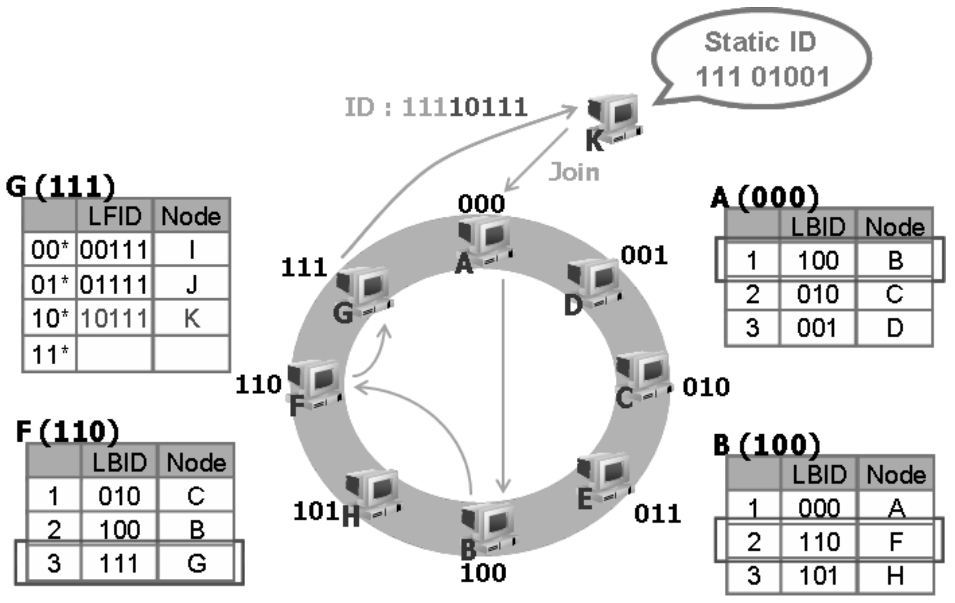
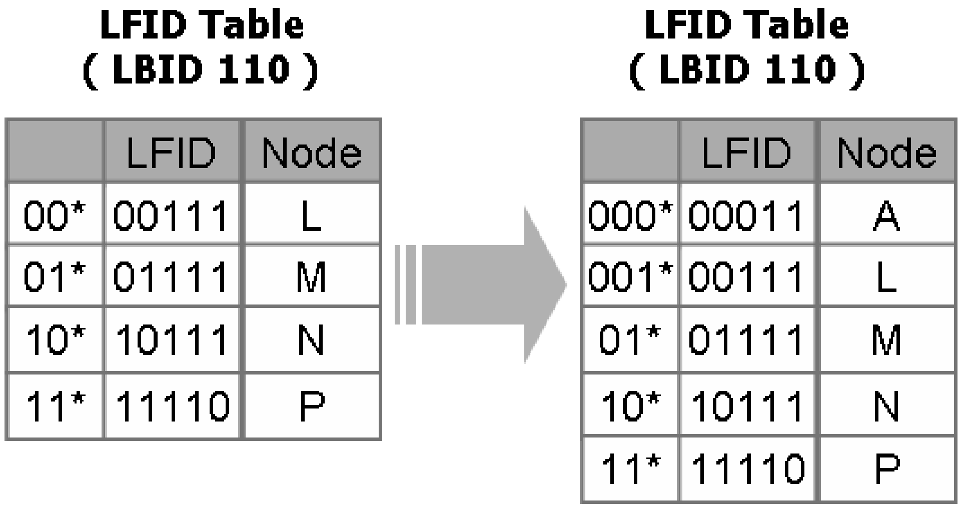


Figure 4.9: Routing of join request and LFID assignment

for the ID space from the prefix to the LFID for each slot. The node L is responsible for servicing the objects whose LFID of their keys are from 00000 to 00111. The node M is responsible the objects whose LFID of their keys are from 01000 to 01111. The node P has 11110 as its LFID. As the rule of LBID assignment, the representative node of this subregion has 11011111 as its nodeID. If LFID of P is 11111, this nodeID conflicts with the nodeID of the representative node. So, the last slot of a LFID table has 11110 LFID to avoid the nodeID confliction.

Unlike the LBID assignment, a new node uses the unique hashed value which is the static nodeID to distribute the transient nodes uniformly and to balance the number of the transient nodes for each sub-region. The figure 4.9 shows the basic LFID assignment. When a new node joins, it sends the join request to any one of participant nodes. This join request routes to the representative node whose LBID is the same to the LBID of the unique node



LFID Table (LBID 110)

	LFID	Node
00*	00111	L
01*	01111	M
10*	10111	N
11*	11110	P

LFID Table (LBID 110)

	LFID	Node
000*	00011	A
001*	00111	L
01*	01111	M
10*	10111	N
11*	11110	P

Figure 4.10: Extension of LFID table

key which is generated by hashing the unique identifier of the new node. This routing process is the most prefix matched method which is generally used by other p2p algorithms. The target representative node which gets the join request considers the new node as the transient node and assigns the LFID to it with the same LBID. To help assigning LFID, every representative node has the LFID slot which divides the sub-region and each slot manages the transient node information. In figure 4.9, the new node, K, joins and its static nodeID is 11101001. At first, it sends join request to node B and B route this message to node F by using the LBID routing and LBID 111. Finally, the join request is forwarded to node G whose LBID is 111 and node G check its LFID table. In this case, the slot with 10* prefix is empty and node K is assigned at this slot. Finally, the new node K gets new nodeID, 11110111, and joins at this system.

Sometimes, many transient nodes are concentrate on a subregion and there is no empty LFID slot. In this case, LFID table should be extended. Like figure 4.10, when the new node A joins and there is no empty LFID slot, its

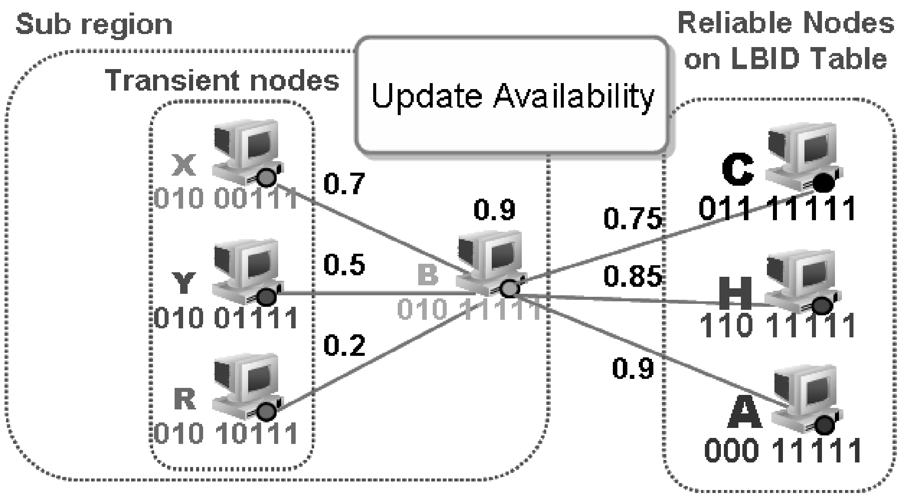


Figure 4.11: Node availability update

first LFID slot is divided by two slots. That is, 00* prefix is divided by 000* prefix and 001* prefix. In this case 001* prefix is already assigned to node L and 000* prefix is assigned to new node A. This LFID table is flexible and it can manage various number of transient nodes.

According to this behavior, the most nodes which join/leave frequently act as the transient nodes and the behavior-aware p2p protocol can reduce the compulsory maintenance overhead by assigning the small ID region to them.

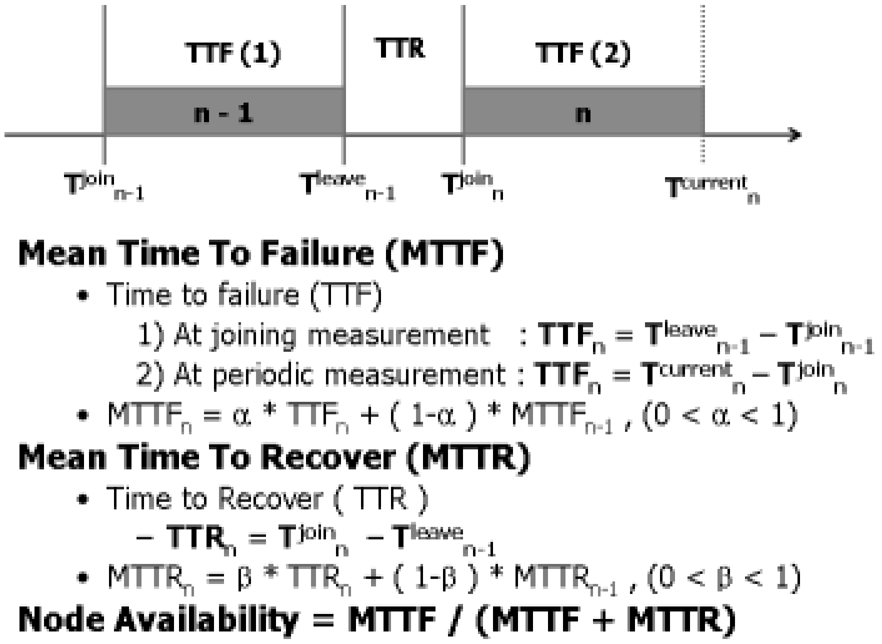
4.3 Data Management

The general servers have very high availability and the data management on them is more simple and there is few overhead. However, in the p2p systems, unlike the general servers, the participants have very low availability. In this case, to preserve the availability of data, there are many replications for the data. These data are the basic p2p system information such as routing information and node information and the object data which is managed by each node which is responsible for some ID region. Previous DHT based p2p systems manage the replication by using the sequential node list such as the successor list for the chord and the leaf set for the pastry. This approaches take too much overhead to preserve the replications when the join/leave occurs frequently.

In the behavior-aware p2p protocol, each representative node knows not only the the availability of itself, but also the availabilities of the other nodes such as transient nodes and LBID routing entry nodes which is managed by the representative node like figure 4.11. In this case, the plentiful information of the availabilities is exploit to preserve the data availability of each subregion above the target availability and reduce the data management traffic according to the dynamic membership. That is, more available nodes replicates data, more data traffic the p2p system can reduce when the nodes frequently join/leave. In figure 4.11, the representative nodes B for the subregion 010 knows its LBID routing table and its transient nodes. Basically, the routing entries are representative nodes and they have more available than other transient nodes. However, these representative nodes are mainly responsible for each subregion and take many jobs. So, in each subregion, the representative node selects the replicas among the transient nodes preferentially, then the routing entries are selected. In this case, the transient nodes which has very few availability such as node R in figure 4.11 are not selected as replicas.

4.3.1 Availability Prediction

I assume that the availability is the prediction value how long a node is alive, because in other researches[11][14], the long lived nodes generally have the large bandwidth and the big computing power. The figure 4.12 shows this availability prediction mechanism. The *Mean Time To Failure* and the *Mean Time To Recover* are used to estimate the node availability. MTTF is the average value how long a node is alive after it joins and MTTR is the average



Mean Time To Failure (MTTF)

- Time to failure (TTF)
  - 1) At joining measurement : $\mathbf{TTF_n = T^{leave}_{n-1} - T^{join}_{n-1}}$
  - 2) At periodic measurement : $\mathbf{TTF_n = T^{current}_{n} - T^{join}_{n}}$
- $MTTF_n = \alpha * TTF_n + (1-\alpha) * MTTF_{n-1}$, $(0 < \alpha < 1)$

Mean Time To Recover (MTTR)

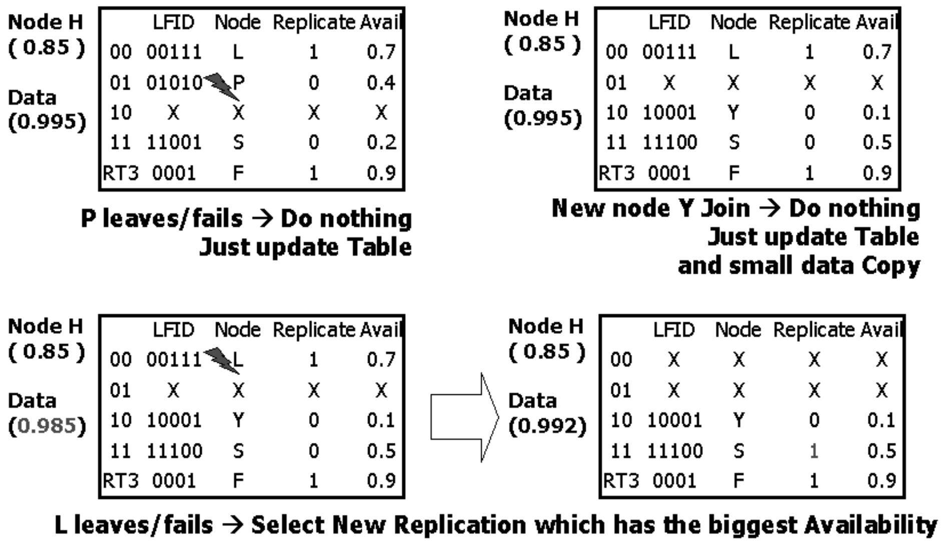
- Time to Recover (TTR)
  - – $\mathbf{TTR_n = T^{join}_{n} - T^{leave}_{n-1}}$
- $MTTR_n = \beta * TTR_n + (1-\beta) * MTTR_{n-1}$, $(0 < \beta < 1)$

Node Availability = MTTF / (MTTF + MTTR)

Figure 4.12: Node availability prediction

value how long a node is sleep after it leaves. TTF and TTR are obtained by using the last join time, the last leave time and the current join time. Unlike TTR, the TTF is periodically updated by using the current time and the current join time. The MTTF and MTTR is obtained by the weighted average of TTF and TTR. The node availability is computed with the equation, MTTF/(MTTF + MTTR). Any node gets its availability by using this process and notify the availability to connected nodes such as a representative node and transient nodes on the same sub-region when it joins the p2p system. Moreover, after the node joins, it periodically estimates its availability and notifies the new value for the fresh information.

Data Availability means the total availability when the multiple nodes have the data. This data availability is obtained by subtracting the probability of that all nodes which have the data leave from 1. That mean if only one node


Node H (0.85)
Data (0.995)

	LFID	Node	Replicate	Avail
00	00111	L	1	0.7
01	01010	P	0	0.4
10	X	X	X	X
11	11001	S	0	0.2
RT3	0001	F	1	0.9

P leaves/fails → Do nothing
Just update Table

Node H (0.85)
Data (0.995)

	LFID	Node	Replicate	Avail
00	00111	L	1	0.7
01	X	X	X	X
10	10001	Y	0	0.1
11	11100	S	0	0.5
RT3	0001	F	1	0.9

New node Y Join → Do nothing
Just update Table
and small data Copy

Node H (0.85)
Data (0.985)

	LFID	Node	Replicate	Avail
00	00111	L	1	0.7
01	X	X	X	X
10	10001	Y	0	0.1
11	11100	S	0	0.5
RT3	0001	F	1	0.9

Node H (0.85)
Data (0.992)

	LFID	Node	Replicate	Avail
00	X	X	X	X
01	X	X	X	X
10	10001	Y	0	0.1
11	11100	S	1	0.5
RT3	0001	F	1	0.9

L leaves/fails → Select New Replication which has the biggest Availability

Figure 4.13: Data Replication Set and its operation

is alive, the data is available.

4.3.2 Data Replication Set

Each representative node is responsible for managing the LBID routing table and storing the objects which is in the sub-region. To keep the data availability of the stored information over the target availability with small maintenance cost, the representative node manages the data replication set which is composed of more available nodes among the transient nodes and the nodes of the LBID routing entries. The figure 4.13 shows the operation of the data replication and its target availability is 0.99. In this figure, Node H (0.85) means its availability is 0.85 and Data (0.995) means its data availability is 0.995. When the transient node P leaves, the representative node just updates the LFID slot, and when the transient node Y joins, the representative node just updates the LFID slot and copies the data of new id range which the node Y is responsible for. That is, because these transient nodes hardly become a member of the replication set, the joins/leaves of them make little management cost. However, when the node L which is the one of the data replication set leaves and the total availability decreases below the threshold value($0.985 < 0.99$), it should select a new replication to keep the high data availability. It does not select the node with the highest availability , but computes the estimated data availability with the every node in the order of the node availability until the estimated value is bigger than the target value. In this example, it first tries Y but this node is in no condition to keep the data availability, and it selects S because of the computed data availability is bigger than the target availability($0.992 > 0.99$). This behavior prevents that the replications converge on few reliable nodes.

4.3.3 Candidate Node

When the reliable node creates the data replication set, it selects more available nodes among the transient nodes and routing entry nodes. In this case, most of the replicate nodes are the routing entry nodes which are the reliable nodes. However, these reliable nodes do too much jobs and the behavior-aware p2p protocol should prevent the additional overhead for them. Moreover, when the reliable nodes leave or fail, the alternative nodes which replace the left or failed reliable node with fast response time are needed.

To achieve these, some transient nodes for the data replication set are selected. These more available transient nodes is called the candidate nodes. These candidate nodes are included in the data replication set, and replicate the routing information such as the LBID routing table and the LFID slot and the data which is in the responsible range of the reliable node. When the reliable node leaves or fails, the most available candidate node replace it by changing transient nodeID to reliable nodeID. This make the failure recovery process easier and faster.

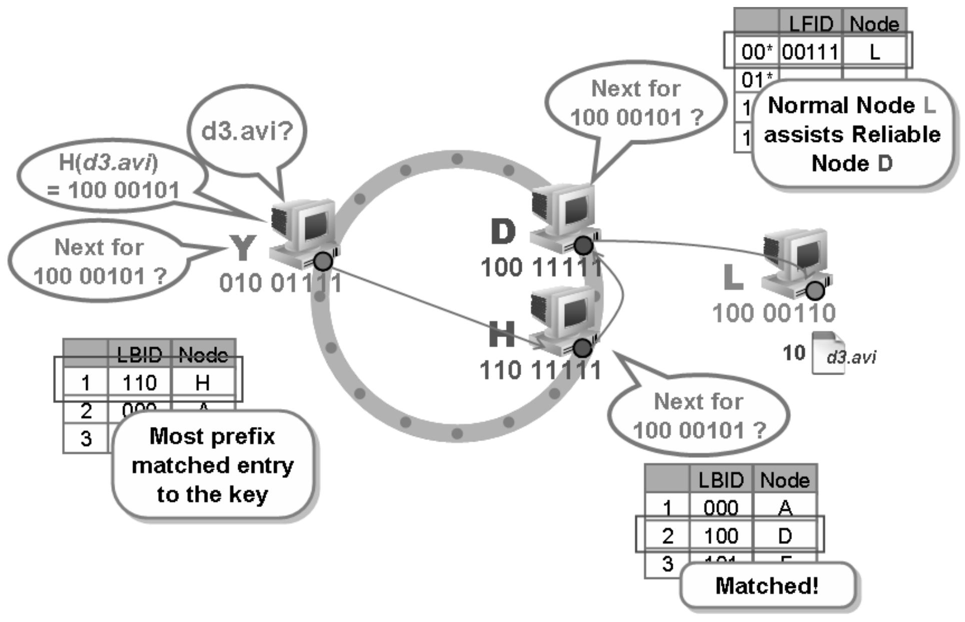


Figure 4.14: Lookup Operation

4.4 Lookup

The lookup process of the behavior-aware p2p protocol is similar to the DHT base p2p protocol. The figure 4.14 shows the simple example of the lookup operation. The normal node Y wants to find ”d3.avi” file and the static object key of d3.avi is 10000101. First of all, node Y check its LBID routing table with the object key 10000101. If the LBID of the object key is matched to the node Y, node Y tries to find the right LFID slot from the LFID table. However, in this case, the LBID of the object key is different to node Y and node Y finds the next routing node from the LBID routing table. This selection is same to the DHT base p2p and the node selects a next routing node as the most prefix matched entry to the object key. In this figure, the next routing node is H and H also selects next routing node D by using most prefix matched method.

Finally, lookup request is forwarded to node D whose LBID is 100 which is same LBID of the object key. After finding right subregion, the representative node D checks its LFID table. If the right slot is empty, the representative node gets this lookup request. Otherwise, the representative node forward this lookup request to the transient node which is assigned by the right LFID slot. In this figure, the object key 10000101 is managed by the first LFID slot whose prefix is 00* and the node L is assigned by this slot. So, the lookup request is forwarded to the node L and node L returns the wanted object "d3.avi" to node Y. According to this assistant, the representative nodes lessen their responsible loads.

The lookup operation is composed of the LBID lookup and the LFID lookup. The LBID lookup takes $O(logN_r)$ cost where N_r is the number of the representative nodes and the LFID lookup takes $O(1)$ cost. Generally, N_r is about $0.2 * N$ where N is the number of total nodes because of the characteristics of participant nodes in the p2p systems. Consequently, the total lookup cost is proportional to $O(logN)$ where N is the number of total nodes. That is, this lookup operation is scalable.

4.5 Node Failure Recovery

In the p2p network, the important information is distributed well. In this case, when a node fails, the probability of losing the information increases. To prevent this fault, the p2p protocol support the replication mechanism. In this section, the reaction and the management cost for the failure are described.

4.5.1 Transient Node Failure

The failure of a transient node is detected, when there is no more LFID slot update messages or there is no more responses for a request. This transient nodes reside below the reliable node and only support small portion of id space which is determined by the LFID. The failure of the transient node can not

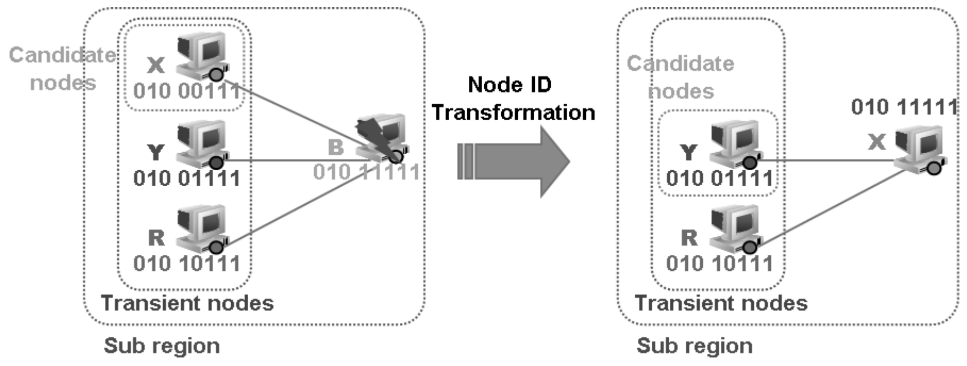


Figure 4.15: Failure Recovery for Reliable Node

affect the important information. According to this features, the LFID slot update is the only management job which needs little traffic.

4.5.2 Candidate Node Failure

The failure of a candidate node is detected with similar conditions of the failure of a transient node, because a candidate node is also a transient node. However, because the candidate node is included in the data replication set, the failure recovery is more complicate. The reliable node updates the LFID slot and choose new candidate node if there is need for replication. When the new candidate node is selected, it replicates data and the routing information.

4.5.3 Reliable Node Failure

The failure of a reliable node is detected, when there is no response for the routing or the updating. However this failure of the reliable and stable node occurs rarely. Because the reliable node do important jobs, the failure recovery should be done as soon as possible. To do this, the candidate nodes are used like figure 4.15. When the reliable node fails, the one of the candidate nodes changes its id nodeID to the nodeID of the failed reliable node. After this nodeID changing, the new reliable node chooses new candidate nodes and updates the routing information and send notification of the new reliable node to the all routing entry nodes. This failure recovery process is more complicate and heavier than the transient node, but the number of the failures is few and this feature prevent increasing the management cost very much.

4.5.4 Update Messages

When churn occurs, affected nodes should update its routing tables such as DHT, LBID table and LFID table to ensure the correctness and efficiency of the lookup process. In general DHT based p2p, when one node joins or leaves, each affected nodes update its DHT. It takes $O(logN)^2$ cost because there are $O(logN)$ affected nodes and it takes $O(logN)$ cost to find one node.

However, in the behavior-aware p2p, the update cost is depend on the type of leaving node. When a transient node joins or leaves, the affected nodes are the representative node and the transient nodes on the same sub-region. Because the transient node is identified by the LFID table, each affected node updates its LFID table only. That is, the update cost is $O(N_t)$, where N_t is the average number of transient nodes for a sub-region. When a representative node joins or leaves, it takes more cost than a transient nodes. This change of a representative node affects the transient nodes on the same sub-region and the representative nodes of its LBID table. Moreover, it also affects the transient nodes of the representative nodes of its LBID table, because every transient nodes contain the same LBID table of its representative node. According to these, the update cost is $O(N_t)*(O(logN_r)+1)$, where N_t is the average number of transient nodes for a sub-region and N_r is the number of representative nodes.

The update cost for representative nodes are bigger than transient nodes. However, the most of churn is caused by the transient nodes. Consequently, the majority of update cost becomes about $O(N_t)$ for the behavior-aware p2p. In this case, if $O(N_t)$ is smaller than $O(logN)^2$, the behavior-aware p2p updates its routing table more efficiently than the DHT based p2p.

5. Performance Evaluation

5.1 Simulation Setup

I make a p2p simulator which emulates the node behavior on the application layer. The previous DHT based p2p systems such as pastry and chord and the behavior-aware p2p systems are implemented. 160 bit ID space is used to identify nodes and the number of LBID bits changes according to the number of the representative nodes which are assigned by the total number of the participant nodes. The total number of nodes are varied from 512 to 8192.

The comparative protocols are follows : DHTP means the DHT based P2P and DHTPA means the naive replication approach with availability value. BAP(1:N_t) means the behavior-aware p2p in which the number of representative nodes are $1/N_t$*(total number of nodes). That is, each representative nodes obtain about N_t transient nodes. For example, when the total number of nodes is 2048, BAP(1:64) has 32 representative nodes and its LBID bit is 5. At the same case, BAP(1:16) has 128 representative nodes and its LBID bit is 7. The table 5.1 shows the variation of LBID for each BAP according to the number of nodes. When N_t is big, the number of representative nodes is small and LBID bit is also small. When the number of node increases, LBID also

Number of Nodes	512	1024	2048	4096	8192
BAP(1:16)	5	6	7	8	9
BAP(1:32)	4	5	6	7	8
BAP(1:64)	3	4	5	6	7

Table 5.1: LBID Variation for each BAP

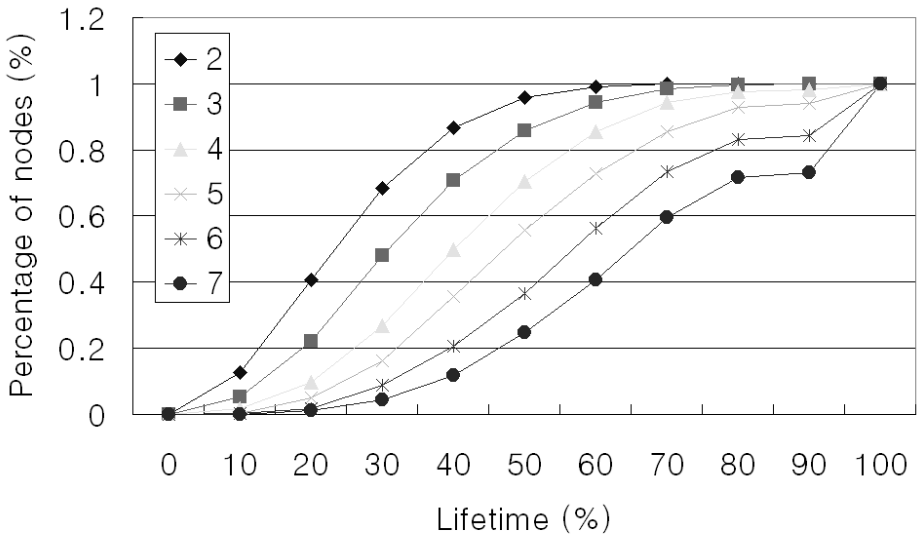


Figure 5.1: Distribution of lifetime with various mean of Poisson distribution

increases to keep the number of transient nodes for each representative nodes.

The value of target availability is 0.999 for the naive replication approach with availability value and the behavior-aware p2p. This value is the same to the 10 replicas for the DHT based p2p where the average node availability is 0.5. That is, the DHT based p2p does not consider the behavior of nodes and it sets the availability of every node to the average value for total nodes.

The participant nodes behave individually and each node has a different lifetime and various join/leave durations. To make the dynamic characteristic, I use the Poisson distribution to identify the lifetime of each node. To assign join/leave duration of a node, the exponential distribution is used. The figure 5.1 shows the lifetime distribution with various mean of Poisson distribution. When the mean is small, most nodes have short average lifetime and frequently join/leave. Otherwise, when the mean is large, most nodes have long average lifetime and they tend to be very reliable. According to this Poisson distribution with 4 of mean value, the lifetime of 80% of total nodes is below 60% of total simulation time, that is, only 20% of total nodes have the reliable server-like profile. Recent researches [11] measure the life distribution of the p2p nodes and show the similar distribution, and I can tell that this distribution is similar to the real world. These characteristics of nodes are assigned when the nodes are created. By using the exponential distribution with these characteristics, I can generate the on-time for which the nodes are on the p2p system and the off-time for which the nodes are off.

5.2 Data Maintenance traffic

5.2.1 Various Number of Nodes

The main problem of the current DHT p2p is the high management cost. In the figure 5.2 and the figure 5.3, the behavior-aware p2p reduces the management cost effectively. To evaluate this cost, I assume that each node obtains same number of objects, that is, if the total number of nodes is 100 and the total number of objects is 100000, and if the total number of nodes is 200, the number of objects is 200000. The number of total nodes is varied from 512 to 8192.

In this case, the behavior-aware p2p reduces the data management cost extremely. The main reason of this improvement is the behavior of transient nodes. In DHT based p2p, the frequent join/leave of transient nodes cause the compulsory copies and update cost for routing information. The naive replication approach with availability value can reduce more traffic than DHT based p2p according to the careful selection of replicas with availability value. However, this naive approach can not ensure efficient use of the reliable nodes whose availability is high and it is also affected by the frequently join/leave nodes. However, in the behavior-aware p2p, the dynamic behavior of transient nodes hardly affects the data availability and this churn can not generate

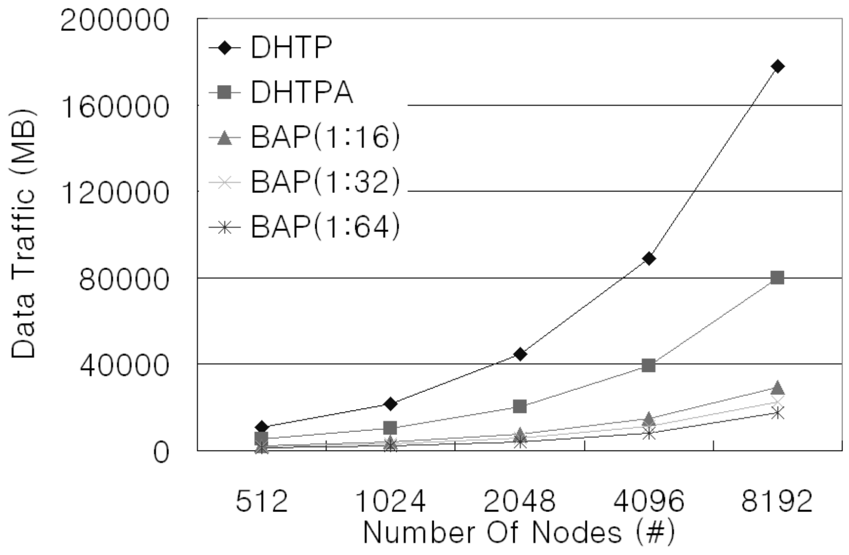


Figure 5.2: Comparison of Total Data Traffic Usage

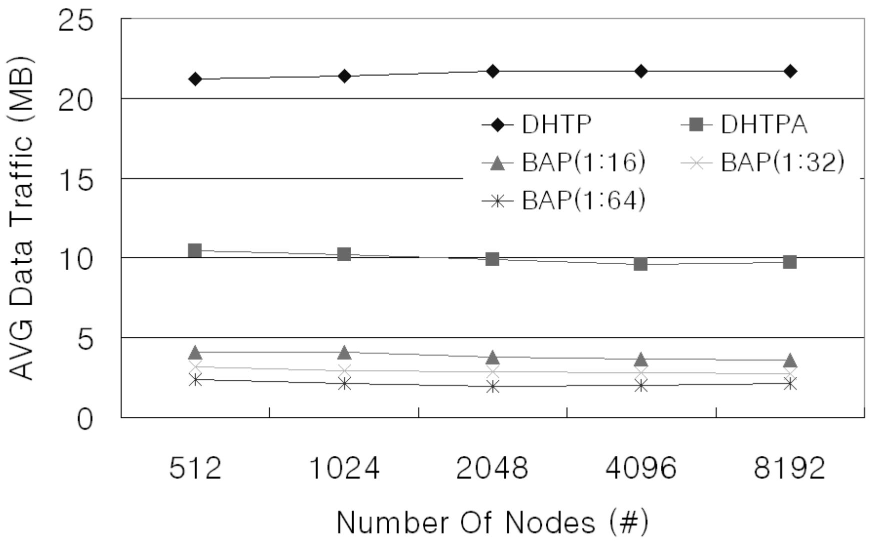


Figure 5.3: Comparison of Average Data Traffic Usage

much data traffic. According to these, the behavior-aware p2p can reduce more management traffic to keep the same level of the high data availability.

One of facts, I should note, is when the number of representative nodes increases, the management traffic also increases. That is, BAP(1:64) reduces more traffic than BAP(1:32) and BAP(1:16). On the same node characteristics, the BAP(1:32) needs more representative nodes than BAP(1:64), and the average availability of the representative nodes of BAP(1:32) is less than BAP(1:64). In BAP(1:32), the transitions for the representative nodes occur more than BAP(1:64) and BAP(1:32) exhausts more network bandwidth than BAP(1:64). According to this fact, when the number of total nodes changes and the system wants to keep its data availability with small maintenance overhead, LBID bits should be changed to the proper number of bits according to the current network state.

5.2.2 Various Lifetime

Figure 5.4 shows the data traffic with various mean of lifetime. The mean of lifetime is the average value of the Poisson distribution. If this mean increases, the average lifetime of nodes also increases. As expected, when the average lifetime of nodes increases, the data traffic decreases in every cases. The data traffic value of the DHT based p2p with mean 7 is greater than any other values of the behavior-aware p2p. It also notes that the behavior-aware p2p can save much more data traffic than the DHT based p2p. In this figure, there is no intersection with each other systems. That is, every system has the similar proportion of decrease of the data traffic according to the increase of the lifetime. Consequently, in the behavior-aware p2p, when the average lifetime increases, there is no jobs to enhance the system. However, if the average lifetime decreases, to keep the average data traffic the behavior-aware p2p changes its LBID to small. For example, when the behavior-aware p2p sets LBID to 7 and works with mean value 7, if the average lifetime decreases

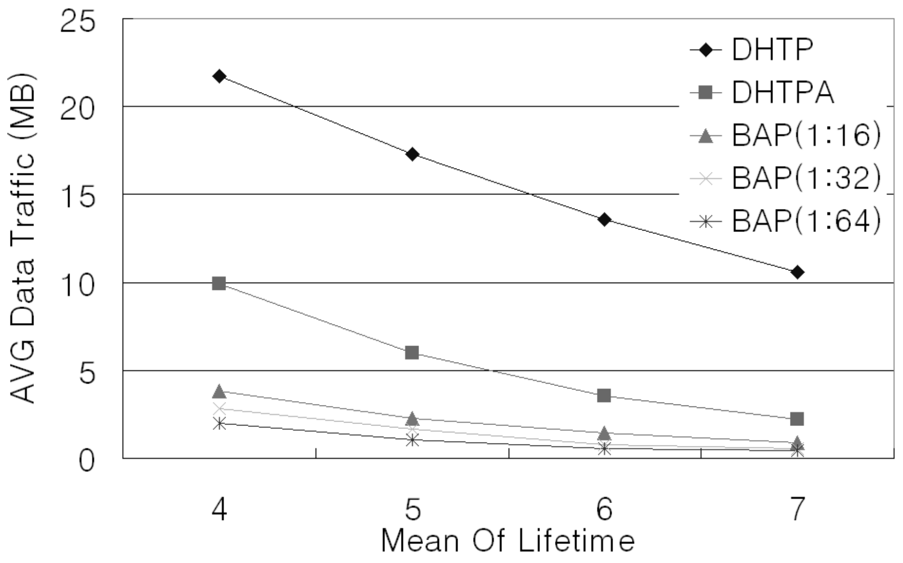


Figure 5.4: Data traffic with various mean of lifetime, node number = 2048

with mean value 6, to keep the data traffic level, the behavior-aware p2p should decrease its LBID from 7 to 6.

5.2.3 Data Traffic Usage in Time Domain

For more detailed inspection for the behavior of the new p2p, I get the data traffic usage in time domain. The figure 5.5 shows the result. The data traffic usage changes according to the behaviors for each time quantum. If many churn occurs for a time quantum, in this quantum, much data traffic is used. In this figure, basically the behavior-aware p2p saves much more data traffic than the DHT based p2p for every time quantum. Moreover, the tendencies of the data traffic for the behavior-aware p2p differs from the DHT based p2p. It means that the behavior-aware p2p can minimize the effect of the churn of the transient nodes. The representative nodes get the main responsibility for the data availability and the transient nodes just assist them. This rule

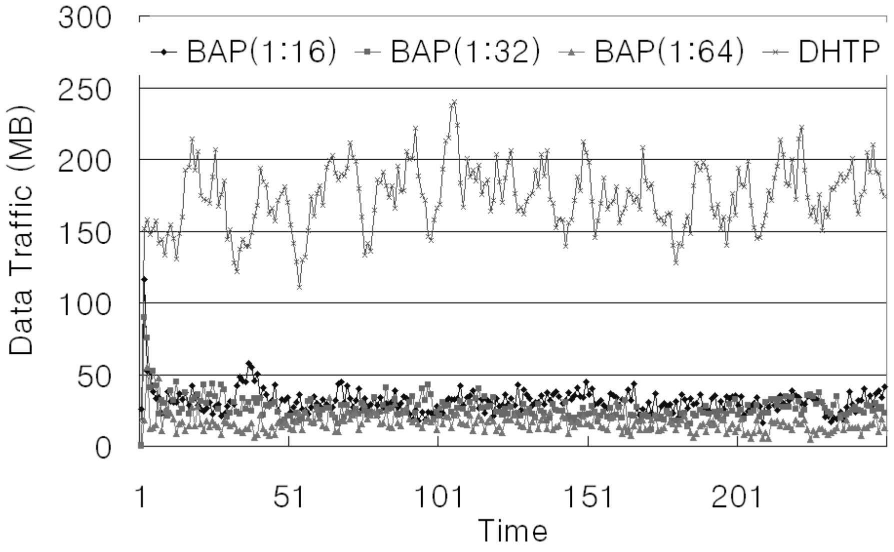


Figure 5.5: Data traffic usage, node number = 2048

makes the transient nodes free from the data copies of the replicas for the data availability under churn. Moreover, the tendencies of one of behavior-aware p2p differ from each other behavior-aware p2ps. The main reason is the different LBID bits. When the LBID bits differ, the number of representative nodes and the number of subregions also differ. According to this difference, the distribution of nodes also differ from each other and the data traffic usages have various tendencies.

The figure 5.6 and the figure 5.7 shows the detailed data traffic usage for join and leave in the behavior-aware p2p. The figure 5.6 shows the join data traffic that means the data traffic usage when a new node joins. In this figure, there is no average level to classify each system. When a new node joins, generally this new node acts as a transient node and there is only small data copy for the LFID slot. Because of this behavior, the average data traffic of join data traffic is small and it is hard to identify the system with the join

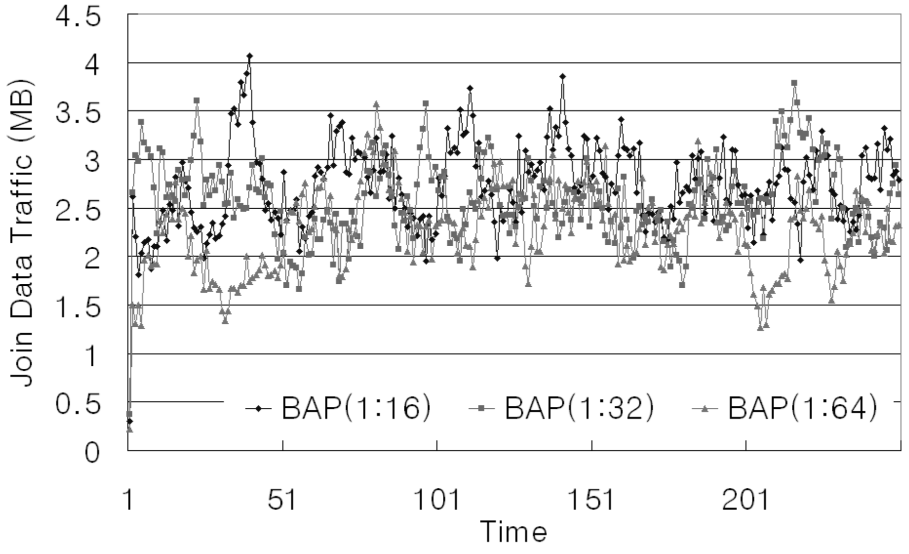


Figure 5.6: Join Data traffic usage, node number = 2048

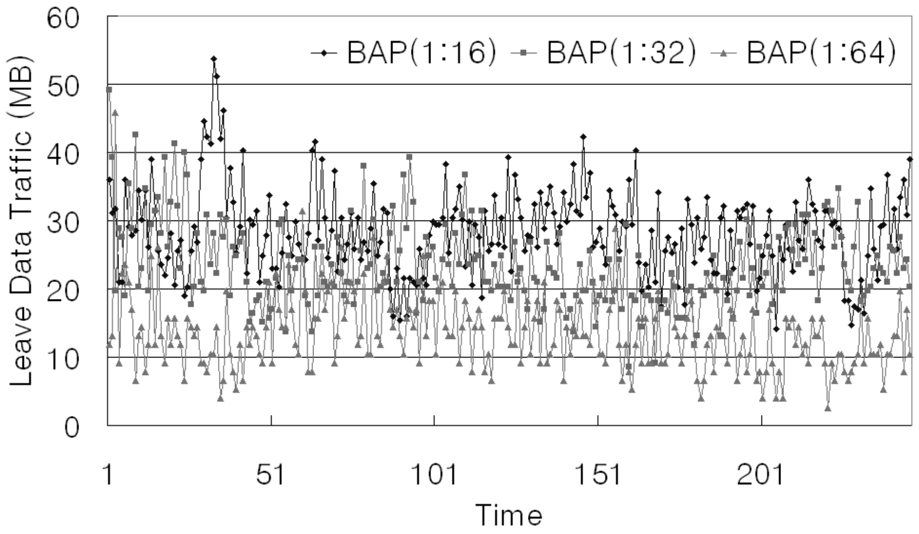


Figure 5.7: Leave Data traffic usage, node number = 2048

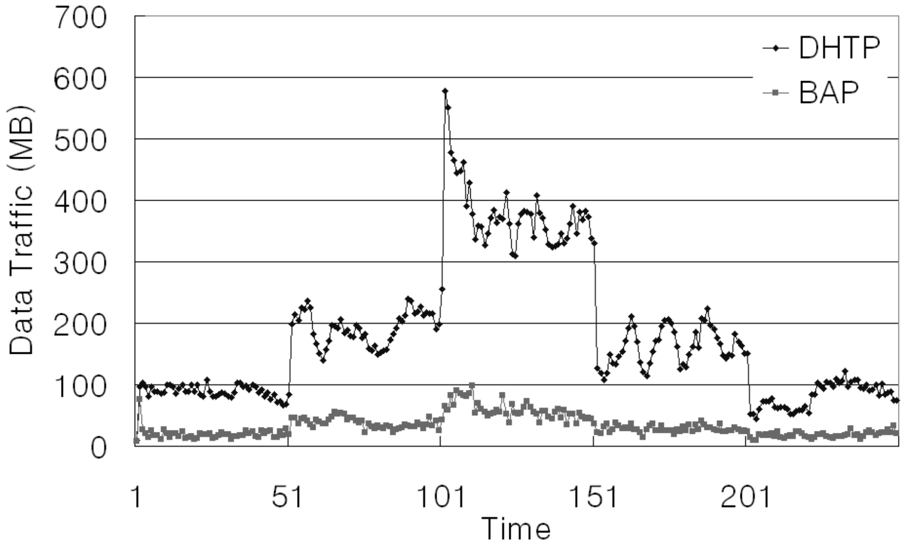


Figure 5.8: Data traffic usage, node number (1024-2048-4096-2048-1024)

data traffic. However, the figure 5.7 shows that the behavior-aware p2ps are easily classified. This figure shows the leave data traffic that means the data traffic usage when a node leaves. In this figure, a behavior-aware p2p with more representative nodes(BAP(1:16)) use more data traffic than a system with less representative nodes(BAP(1:64)). When the system has more representative nodes and more subregions, the average availability of nodes for a subregion decreases. In this case, the possibility with that a node leaves for a subregion increases and each subregion needs more replicas and more data traffic.

5.2.4 Data Traffic Usage with Dynamic Network state

To find out how the behavior-aware p2p follows up the dynamic network state, when the number of total nodes is changed on the fly, the data traffic usage for each p2p system is observed. The figure 5.8 shows the data traffic usage with the dynamic network state. The number of nodes initially 1024 and it

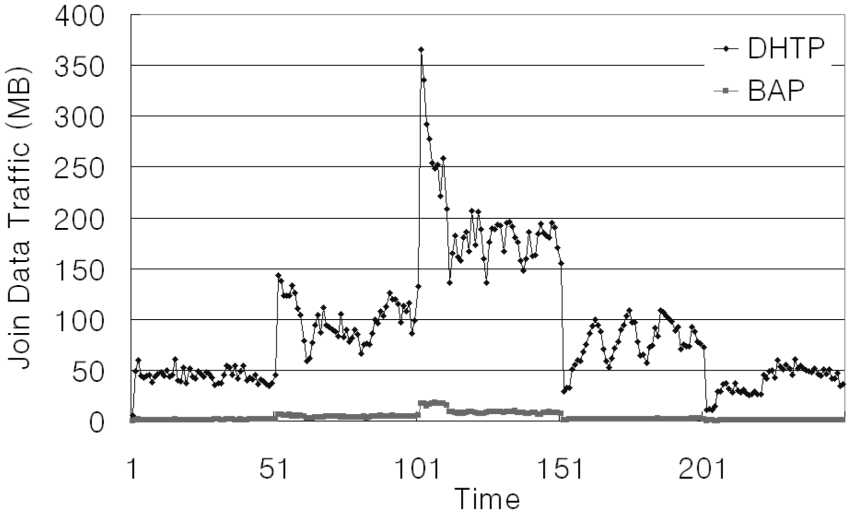


Figure 5.9: Join Data traffic usage, node number (1024-2048-4096-2048-1024)

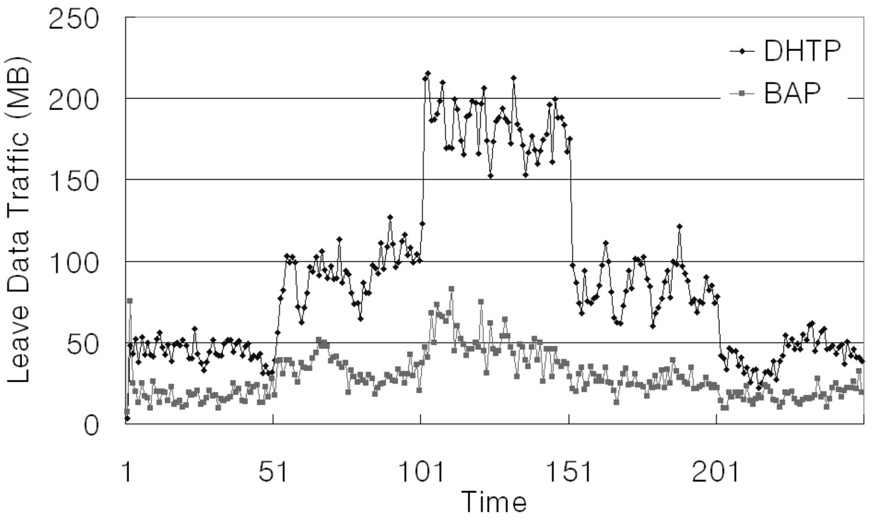


Figure 5.10: Leave Data traffic usage, node number (1024-2048-4096-2048-1024)

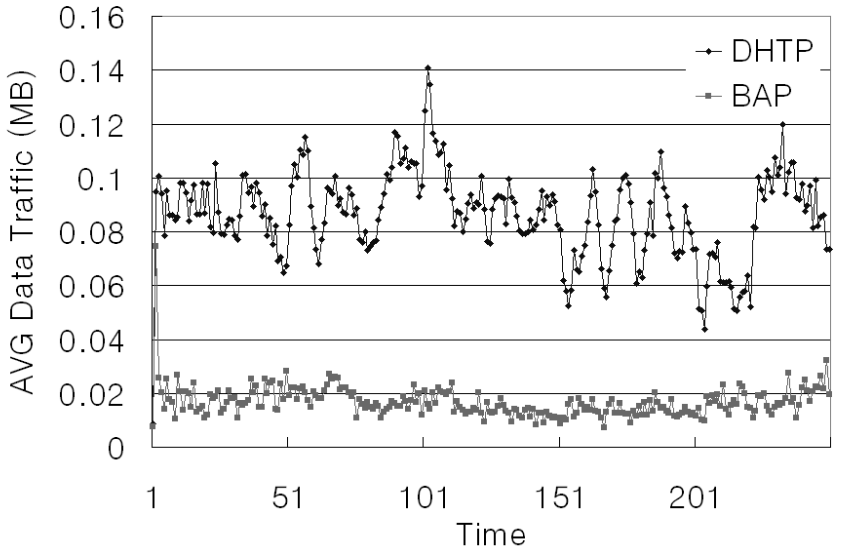


Figure 5.11: Average Data traffic usage, node number (1024-2048-4096-2048-1024)

changes to 2048 at 50. At 100 it increases to 4096 and finally it decreases to 2048 at 150. In this figure, P means the DHT base p2p and BAP means the behavior-aware p2p with 6 bit LBID. This BAP is same to BAP(1:16) when the number of nodes is 1024, BAP(1:32) when the number of nodes is 2048 and BAP(1:64) when the number of nodes is 4096. Basically, BAP reduces more data traffic than DHT based p2p. When the number of nodes increases, the level of data traffic also increases for every system. Especially, at the instant of the change, there is a peak traffic.

For more detailed inspection, I divide the data traffic to the join data traffic and the leave data traffic. The figure 5.9 shows the join data traffic usage. In DHT base p2p, it takes much join data traffic and there is a peak traffic when the number of nodes increase. However, in BAP, the join data traffic is very small even if it increases when the number of nodes increase and there is no peak traffic. It means the normal nodes which frequently join/leave as transient nodes can not exhaust the data traffic. The figure 5.10 shows the leave data traffic usage. According to this figure, It is found out that the main data traffic usage of BAP is caused by the leave data traffic. When the number of nodes increases the leave data traffic also increases.

The figure 5.11 shows the average data traffic of each system. Surely, the average data traffic for the DHT based p2p has no tendency. However, I find out that the average data traffic decreases, when the number of nodes increases for BAP. In this case, according to the result from previous subsection, the average data traffic of BAP(1:16) is bigger than BAP(1:32). So, when the number of nodes increases and the LBID bits does not change, the average data traffic decreases. However, I note that in this case, the load of the representative nodes increases because the number of transient nodes for each representative node increases. When the number of transient nodes increases, the responsible data range and number of objects for a representative nodes also increase. Moreover, because a representative node and other transient nodes of a same subregion should know the same LBID table and LFID table, when the number of transient nodes increase, the update message should be increases. Consequently, when the number of nodes increases, I should consider the tradeoff between the reduction of the average data traffic and the increase of the load of a representative node. According to this tradeoff, the LBID bits are changed to fit the behavior-aware p2p to the current network state.

5.3 Lookup Hops

In the p2p system, the lookup cost is also important parameter for the scalability because there are too many participants. Figure 5.12 shows the comparison of the lookup hops. For all systems, the lookup hops are proportion to the Log

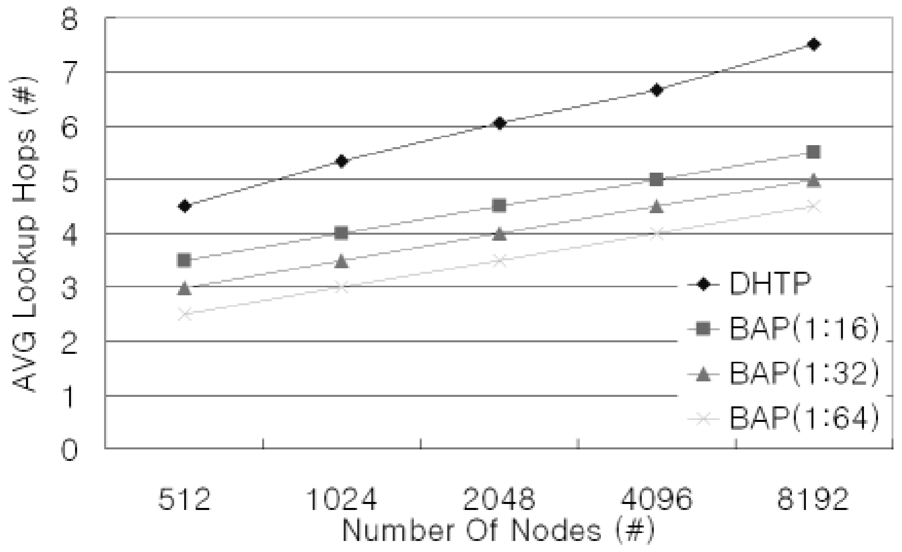


Figure 5.12: Comparison of Lookup Hops

N, where N is the total number of nodes. The behavior-aware p2p performs more efficient lookup than normal DHT. The reason is that the behavior-aware p2p system uses the representative nodes to route the lookup request and the number of these nodes are much less than the total nodes. These representative nodes are more stable and more powerful than other nodes and they are durable nodes for the many routing requests. Additionally, the transient nodes assist the representative nodes to take the request for the ID region and the load of the representative node are reasonable.

In the figure 5.12, the slope of the DHT p2p is steeper than the Slopes of the behavior-aware p2p. In general DHT p2p, the routing table is filled with unreliable nodes. When these nodes leave, many routing entries can contain wrong and failed routing information. This wrong information makes many routing faults and it takes times to route to the right destination. According to this fact, the slope of DHT p2p is steeper. However, in the behavior-aware

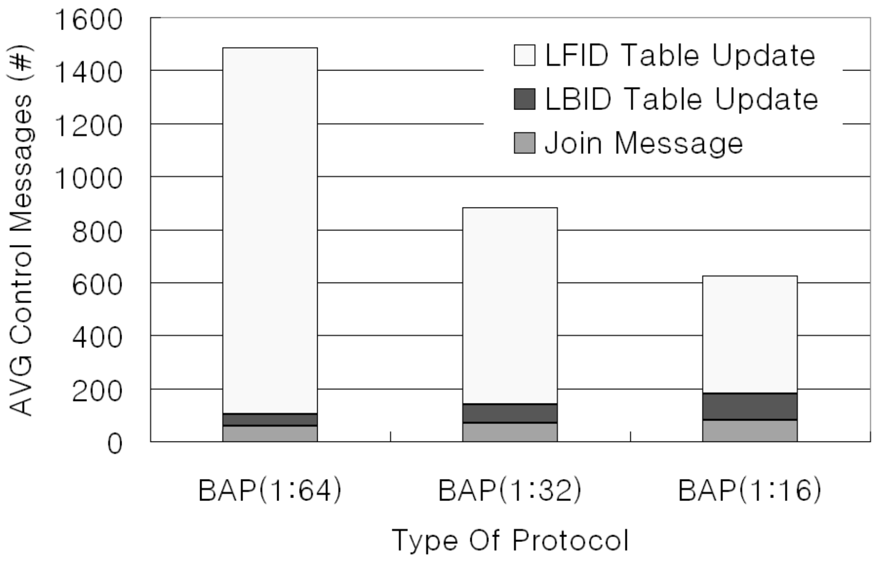


Figure 5.13: Average Control Messages for each BAP, node number = 2048

p2p, every routing entries are filled with reliable nodes and there are very few routing faults. This make the slopes of the behavior-aware p2p gentler than DHT p2p.

Moreover, when the LBID bit is big, the average routing hop is also big. That is, BAP(1:64) has less representative nodes than BAP(1:32) and BAP(1:64) also has less routing table than BAP(1:32). According to this small routing table, BAP(1:64) needs small routing hops than BAP(1:32).

5.4 Control Traffic

Figure 5.13 shows the needed control messages for the various behavior-aware p2p. There are 3 types of control messages: LFID table update, LBID table update and Join Messages. LFID table update is affected by the number of the transient nodes for a representative node. LBID table update and Join messages are affected by the number of the representative nodes. When the number of transient nodes increase, LFID table update increases. As the same way, when the number of representative nodes increase, LBID table update and Join messages increases.

In this figure, BAP(1:64) needs most control messages and BAP(1:16) uses least messages. This is because the majority of the control messages for the behavior-aware p2p is the update for the LFID table. BAP(1:64) keeps the average number of transient nodes in about 64 and in BAP(1:16), the average transient nodes for a representative node is about 16. That is, BAP(1:64) has 4 times more transient nodes than BAP(1:16). On the other side, BAP(1:16) has 4 times more representative nodes than BAP(1:64). According to this, the average number of control messages for the LFID table is about 4 times more than BAP(1:16). Consequently, even if in BAP(1:64) LBID table update and join messages are less than BAP(1:16), BAP(1:64) needs about 2.5 times more control messages than BAP(1:16).

To find out the more characteristics of the control messages, figure 5.14 and figure 5.15 show the average control messages on various number of nodes and various lifetime. Generally, behavior-aware p2p uses less control traffic than DHT based p2p. As described, BAP(1:64) uses more control traffic than other BAPs. When the number of nodes increases, the average control messages for any p2p increase. However, in figure 5.15, on various lifetime, the behavior-aware p2p is not always better than DHT based p2p. When the mean of lifetime increases, the average control messages for DHT based p2p decreases. But, even if BAP(1:16) decreases the number of control messages and BAP(1:32) preserve the same level of the average control messages, BAP(1:64) increases the control messages more than DHT based p2p. The main reason is the overprovision for the number of transient nodes. That is, when the mean of lifetime increase, the number of reliable nodes which have a server-like profile increases and in BAP(1:64), many reliable nodes acts as transient nodes. In

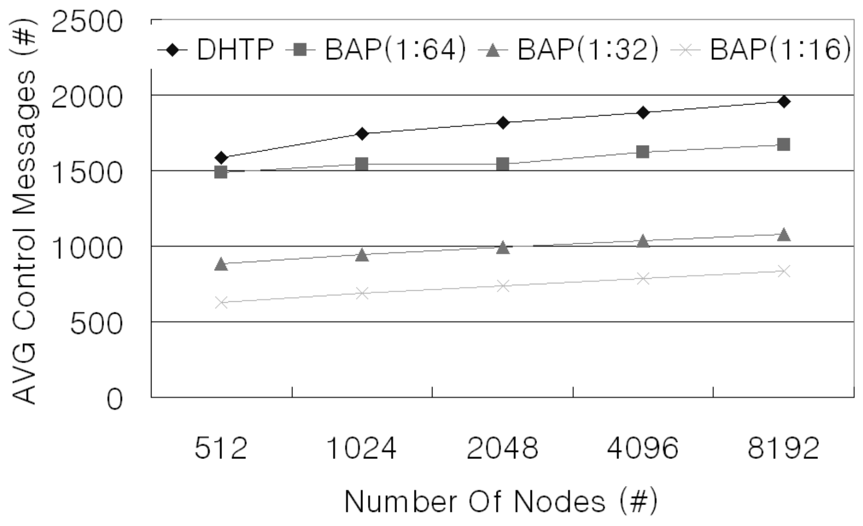


Figure 5.14: Comparison of Average Control Messages for various number of nodes

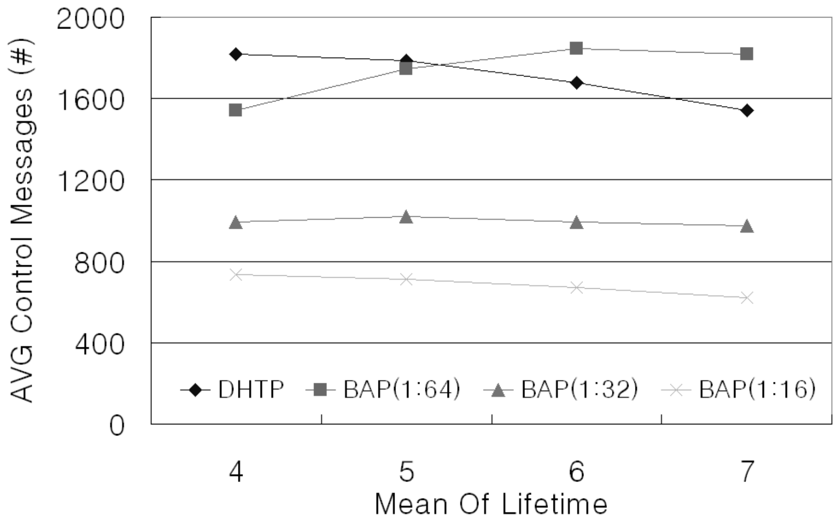


Figure 5.15: Comparison of Average Control Messages for various mean of lifetime

this case, the number of transient nodes which are affected by churn of the normal and weak nodes increases and LFID table update increase enormously. To prevent this compulsory control traffic, the behavior-aware p2p increases its LBID and has more representative nodes.

5.5 Load Balance

The figure 5.16 shows the lookup distribution for the total nodes. In this figure, I define the lookup load of a node as the number of lookup requests of it divided by the average number of lookup requests of whole nodes. As the nature of the previous DHT based p2p, the load is distributed to the whole of nodes by the shape of the normal distribution and the average load of nodes is nearly 1. This behavior causes the heavy information maintenance overhead because the nodes which join/leave very frequently can be responsible for the big ID region. On the other hand, in behavior-aware p2p, the load distribution can be classified into the representative nodes and the transient nodes. About 75 percent of nodes have less load than other nodes because these nodes act as transient nodes which join/leave frequently and they takes the responsible for small ID region which is assigned by the LFID. The average load of the transient nodes is about 0.4 and these nodes are distributed uniformly. Otherwise, the representative nodes take much more load because they are alive for a long time and represent for the sub-region. The average load of these nodes is about 2. Moreover, when the number of representative nodes increases, the average load of transient node increases. The main reason of this result is that when the representative nodes increase, the transient nodes get more chances to help the representative nodes. That is, the transient nodes act as candidate nodes and get more jobs to help the representative nodes.

The representative nodes get more loads than the transient nodes and it needs to balance the load of the representative nodes for the fairness. The

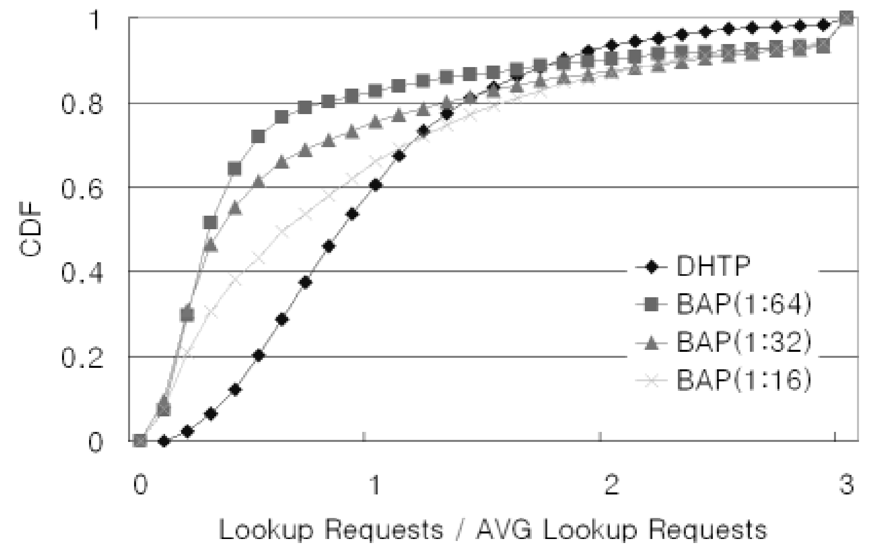


Figure 5.16: Lookup distribution for the total nodes

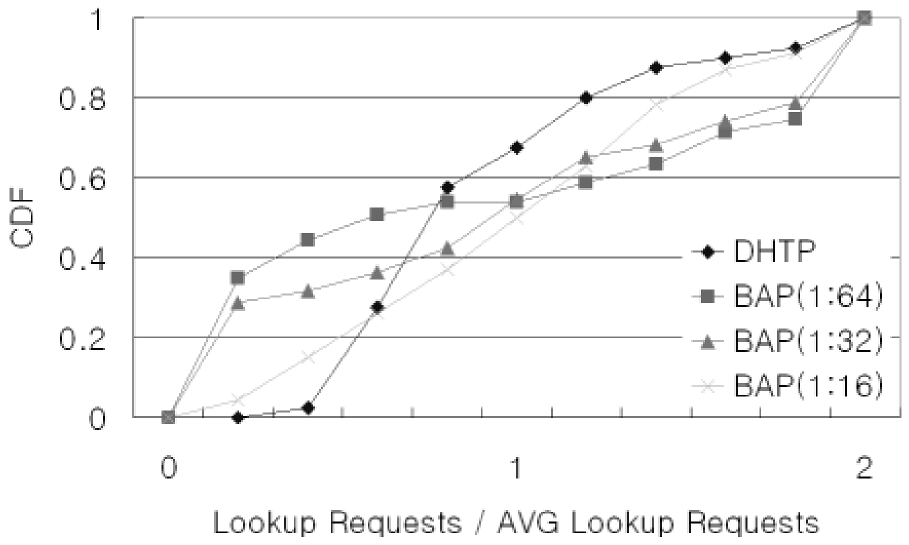


Figure 5.17: Lookup distribution for the representative nodes

figure 5.17 shows the load distribution for the reliable nodes which are about 20 percent nodes of the whole participant nodes. The behavior-aware p2p system balances the load more than the previous DHT based p2p because in the behavior-aware p2p the sub-region is distributed evenly and each reliable node acts as a representative node to be responsible for the sub-region. Some jitters are appeared because the transient nodes assist the representative nodes and the transitions for the representative nodes occur. Moreover, BAP(1:16) distributes the load more uniformly than BAP(1:64) and it means that if the system estimates the number of reliable nodes well, each reliable node takes well balanced load.

The figure 5.18, 5.19, 5.20, 5.21 show the data load distribution of each nodes when the total number of nodes is 2048 and the mean of lifetime is 4. In these figures, X-axis means the nodes that are arranged by its lifetime. That is, on X-axis, the right side means reliable nodes with long lifetime and the left side means normal nodes with short lifetime. Y-axis means the data traffic that is used by each node. The figure 5.18 shows the load distribution of each node in DHT p2p. There are not any relation between the data traffic and the lifetime. Some normal nodes exhaust more data traffic than other reliable nodes. Each node uses very big data traffic too. Otherwise, the figures 5.19,5.20,5.21 show the relation between the data traffic and the lifetime. In the behavior-aware p2p systems, more reliable nodes get more data traffic and normal nodes get less data traffic. That is, the reliable nodes act as representative nodes or candidate nodes and they take data traffic when the churn occurs. Moreover, when the number of representative nodes increases, normal nodes get more data traffic. That is, in BAP(1:16), normal nodes copy more data for data availability than BAP1:64. The main reason is that normal nodes can become candidate nodes and they take some data traffic when the churn occurs. However, in BAP(1:16) the peak data traffic for a representative node is smaller than BAP(1:64). It means that BAP(1:16) has

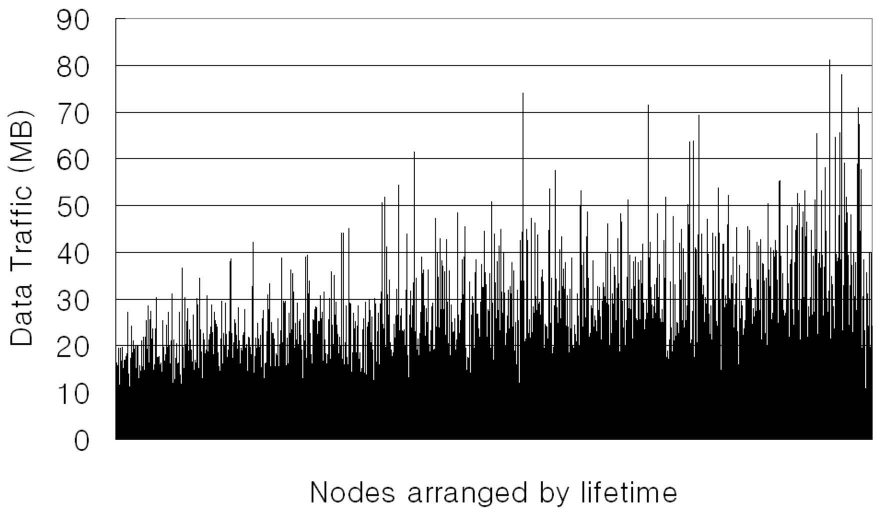


Figure 5.18: Data traffic distribution of each node, (pastry, 2048), mean of lifetime = 4

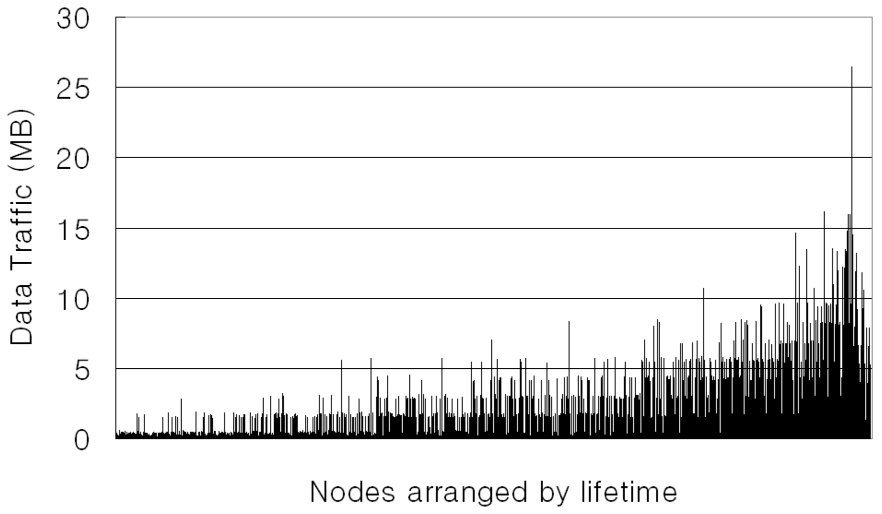


Figure 5.19: Data traffic distribution of each node, (BAP(1:64), 2048), mean of lifetime = 4

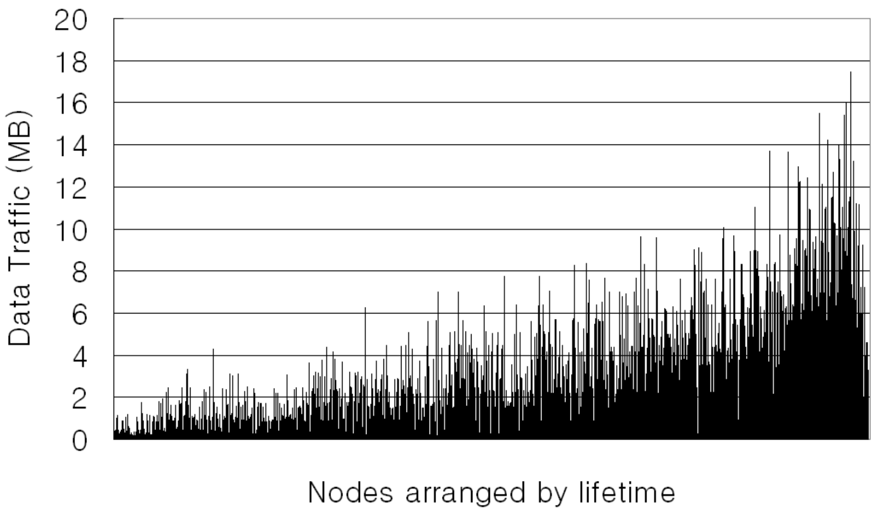


Figure 5.20: Data traffic distribution of each node, (BAP(1:32), 2048), mean of lifetime = 4

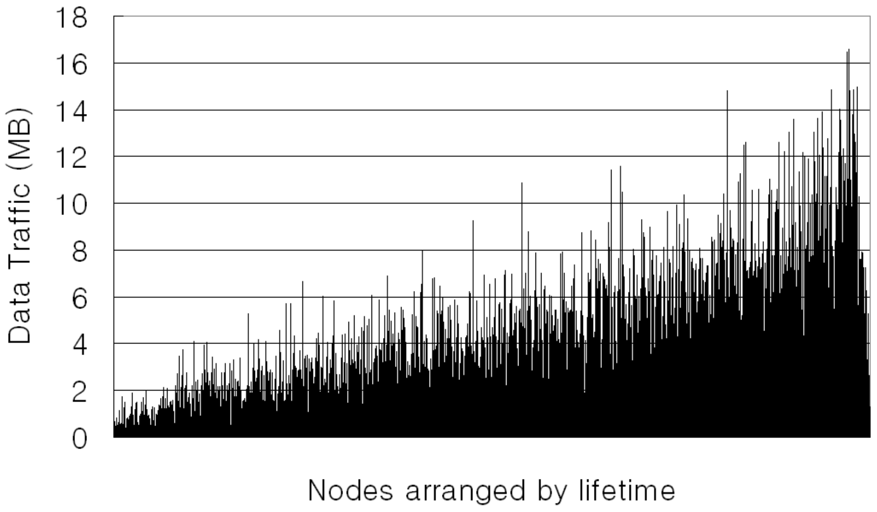


Figure 5.21: Data traffic distribution of each node, (BAP(1:16), 2048), mean of lifetime = 4

more well balanced representative nodes than BAP(1:64) and representative nodes on BAP(1:16) have less works than BAP(1:64).

The figure 5.22, 5.23, 5.24, 5.25 show the data load distribution when the mean of lifetime is 7. In this case, there are more reliable nodes than the previous situation and the average data traffic usage decreases. Like the previous results, in DHT based p2p, there is no relation between the data traffic and lifetime of nodes and in the behavior-aware p2p, the reliable nodes take more data traffic than the normal node. However, in BAP(1:16), the tendency of the data traffic is similar to the DHT based p2p. This is because BAP(1:16) has more representative nodes than BAP(1:64) and the many of transient nodes of BAP(1:16) are reliable nodes. That is, the characteristic of BAP(1:16) is similar to DHT based p2p which considers p2p system as the homogeneous system. However, though the shape of the data traffic usage is similar to each other, the average data traffic of BAP(1:16) is much less than DHT based p2p. Consequently, when the p2p system has more reliable nodes than normal nodes, the behavior-aware p2p increase its LBID to balance the load of the participant nodes.

This feature that classifies the load according to the characteristics of nodes is very useful for the p2p system on the heterogeneous network that is consist of the various nodes such as servers, workstations and PCs. The behavior-aware p2p system can exploit these powerful components efficiently and easily because the server-like nodes locate for the representative nodes automatically.

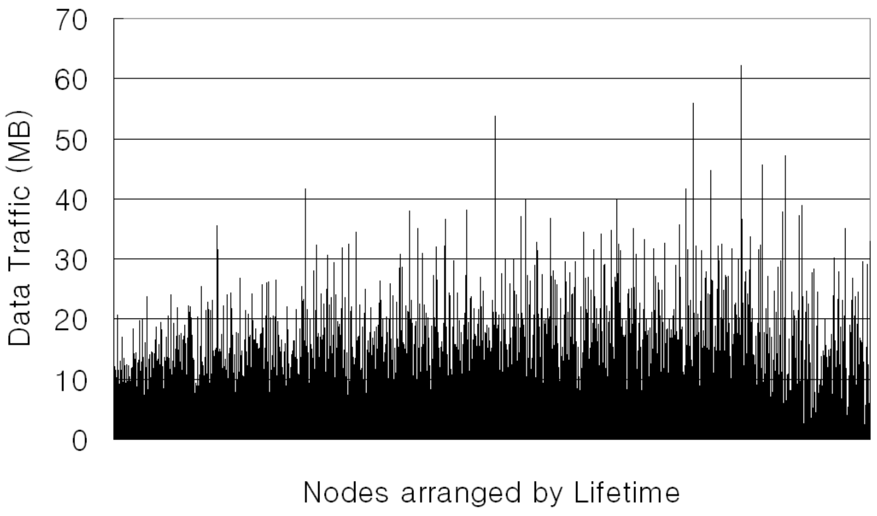


Figure 5.22: Data traffic distribution of each node, (pastry, 2048), mean of lifetime = 7

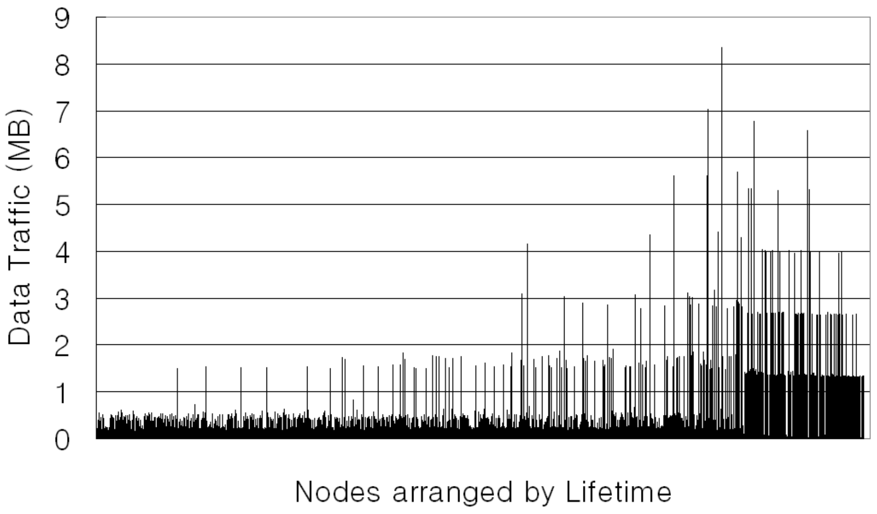


Figure 5.23: Data traffic distribution of each node, (BAP(1:64), 2048), mean of lifetime = 7

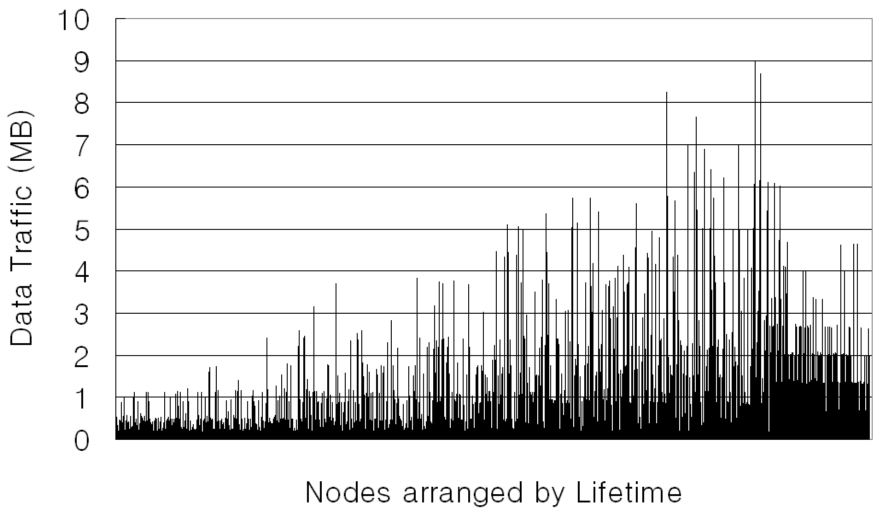


Figure 5.24: Data traffic distribution of each node, (BAP(1:32), 2048), mean of lifetime = 7

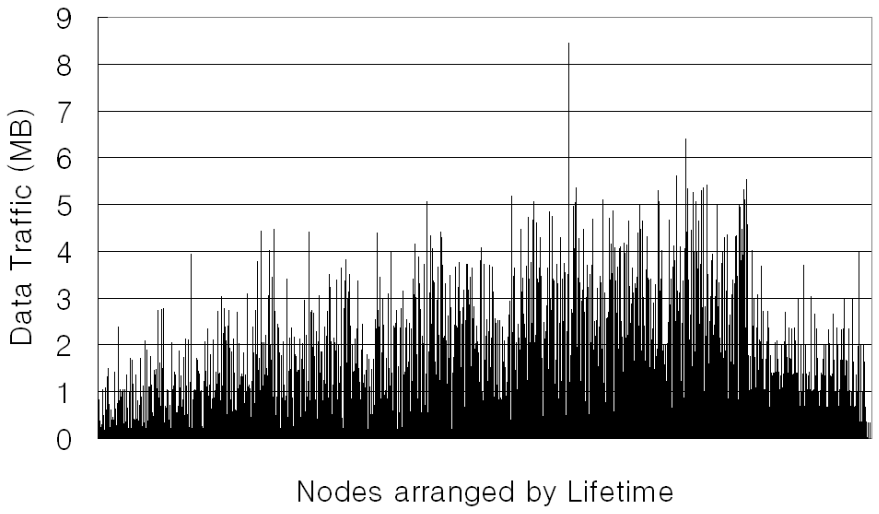


Figure 5.25: Data traffic distribution of each node, (BAP(1:16), 2048), mean of lifetime = 7

6. Conclusions

This dissertation suggests the behavior-aware p2p protocol for high data availability to reduce the information maintenance overhead by exploiting the heterogeneity of participant nodes efficiently. Unlike the DHT based p2p, the nodeID of a node changes on the fly according to its behavior on the p2p system to support the p2p system efficiency and each nodes takes the different responsibility in accordance with its nodeID. The representative node which is the more reliable and more powerful node acts as the more important role such as the routing and the replication. The transient node which joins/leaves very frequently acts as the simple role to reduce the information maintenance traffic. The Load-Balanced ID identifies the representative nodes to balance the loads and the Load-Free ID identifies the transient nodes to reduce the responsibility and eliminate the compulsory maintenance overhead.

This behavior-aware p2p is very good for the p2p system on the heterogeneous environment which is consist of the various kinds of nodes such as servers, workstations and PCs. The server-like nodes are located at the position for the representative nodes automatically and the other nodes act as the transient nodes. This dynamic change of the positions of each node can help exploiting the more reliable and more powerful nodes efficiently and easily. However, this p2p protocol may over-provision for the representative nodes and this may decreases the performance of the p2p system. To compensate this lack, the adaptive method for the whole state of nodes to keep the proper number of representative nodes is needed.

요 약 문

높은 데이터 유효성을 보장하기 위한 행동-인지 피어-투-피어 프로토콜

오늘날 많은 연구 논문들에서 Distributed Hash table(DHT) 기반의 피어-투-피어 시스템에 대한 연구들이 진행되어왔다. 이들 논문들은 수많은 클라이언트로 이루어진 시스템에서 잉여 자원을 효율적으로 모으고 사용할수 있도록 하는 프로토콜에 대해서 주로 연구해왔다. 하지만 피어-투-피어 시스템은 아주 다양한 특성을 가지는 여러 노드들로 이루어져 있고, 이에 따른 각 노드들의 빈번한 참여/탈퇴에 따라 다른 참여 노드들이 영향을 받게 되고, 이러한 영향에 따른 DHT 프로토콜의 성능을 유지 하기 위한 정보 관리 트래픽이 매우 심각하게 발생하게 된다. 더욱이, 기존의 연구들은 주로 피어투피어 시스템에서의 효율적인 데이터 룩업에 집중한 반면, 데이터의 유효성을 효율적으로 보장하는것에 대해서는 소홀하였다.

본 학위 논문은 높은 데이터 유효성을 보장하기 위한 행동-인지 피어-투-피어 프로토콜을 제안한다. 이 프로토콜을 사용함으로써, 피어-투-피어 시스템은 참여 노드들의 다양한 행동을 인지하고 효율적으로 노드들을 사용함으로써 정보 관리 트래픽을 줄일수 있다. 기존의 DHT 기반의 피어-투-피어 프로토콜에서는 변화하지 않는 노드아이디를 사용한다. 이 아이디는 각 노드들을 식별할수 있는 유일한 값, 예를 들면 아이피주소나 맥주소와 같은 값들을 해쉬함수를 사용함으로써 얻을 수 있다. 이러한 변화하지 않는 유니크아이디는 프로토콜을 단순하게 작성할수 있는 장점은 있지만, 각 노드들의 특성에 따라 적절한 위치에서 적당한 역할을 하도록 하기 힘들게 된다. 행동인지 피어-투-피어 프로토콜에서는 이러한 변화하지 않는 노드 아이디 대신 변화가 가능한 다이나믹 노드아이디를 사용한다. 이 다이나믹 노드아이디는 Load Balanced ID(LBID)와 Load Free ID(LFID)로 구성된다. LBID는

보다 안정적이고 성능이 좋은 노드들의 역할을 서로 균등하게 할당 받을수 있도록 하고 LFID는 일반적이고 성능이 약한 노드들의 역할을 줄이고 이들 노드들이 빈번하게 시스템에 참여/탈퇴를 할때 불필요한 정보 관리 트래픽을 줄일수 있도록 한다. 이 다이나믹 노드아이디는 임의의 중앙 관리자 없이 각 노드들의 정보만을 가지고 할당 된다. 새로운 아이디는 조인 메세지를 처리하는 노드의 노드아이디와 현재 네트워크 상태에 따라서 적절한 아이디가 할당된다. 새로운 아이디가 할당될때는 그 노드의 특성과는 관계없이 아이디가 할당 되지만, 노드가 피어-투-피어 시스템에 참여하고 있는동안 노드의 동작에 따라서 노드아이디는 변하게 되고, 마침내 적절한 위치와 역할을 하기 위한 노드아이디를 가지게 된다. 보다 안정적이고 성능이 좋은 노드들은 주로 데이터의 유효성을 보장하고 데이터의 룩업의 정확성을 보장하게 된다. 일반적이고 성능이 약한 노드들은 성능이 좋은 노드들을 도와주는 역할을 하고 담당하는 데이터의 영역을 제한함으로써, 이 노드들이 참여 탈퇴를 빈번하게 하더라도 데이터의 유효성과 데이터의 룩업의 효율성에 영향을 거의 주지 않도록 한다.

새로운 행동-인지 피어-투-피어 프로토콜의 성능을 확인하기 위해서 이벤트 드리븐 방식의 시뮬레이터를 제작하였다. 이에 따라, 행동-인지 피어-투-피어 프로토콜은 기존의 DHT 기반의 피어-투-피어 프로토콜에 비해서 높은 데이터 유효성을 적은 정보 관리 트래픽을 가지고 보장한다는것을 알수 있었다. 또한 데이터 룩업도 기존의 DHT 기반의 피어-투-피어와 같이 효율적임을 알수 있었다. 이러한 행동-인지 피어-투-피어 프로토콜은 기존의 DHT 기반의 피어-투-피어 프로토콜을 실제 어플리케이션에 적용할때 가장 걸림돌이 되었던 노드들의 빈번한 참여/탈퇴에 따른 정보 관리 트래픽을 효과적으로 줄일수 있고, 서버와 클라이언트가 동시에 사용되는 환경에서 임의의 관리자의 노력없이 각 노드들의 특성에 따라 자동적으로 알맞은 역할을 함으로써 효율적인 피어-투-피어 시스템을 구성하도록 한다.

References

[1] K.Kim and D.Park. Efficient and Scalable Client Clustering For Web Proxy Cache. *IEICE Transaction on Information and Systems*, E86-D(9), September 2003.

[2] Z. Xu, Y. Hu and L. Bhuyan, Exploiting client cache: A scalable and efficient approach to build large web. *In Proceedings of IPDPS04*, April, 2004.

[3] S.Iyer, A.Rowstron, and P.Druschel. Squirrel: A decentralized peer-to-peer web cache. *In Proceedings of Principles of Distributed Computing'02*, 2002.

[4] J. Kubiatowicz, D. Binder, Y. Chen, P. Eaton and et al., Oceanstore: An architecture for global-scale persistent storage. *In Proceedings of ACM ASPLOS00*, November, 2000.

[5] I.Stoica, R.Morris, D.Karger, M.F.Kaashoek, and H.Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. *In Proceedings of ACM SIGCOMM 2001*, August 2001.

[6] A.Rowstron and P.Druschel. Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. *In Proceedings of the International Conference on Distributed Systems Platforms(Middleware)*, November 2001.

[7] B.Y.Zhao, J.Kubiatowicz, and A.Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *UCB Technical Report UCB/CSD-01-114*, 2001.

[8] S.Ratnasamy, P.Francis, M.Handley, R.Karp, and S.Shenker. A scalable content-addressable network. *In Proceedings of ACM SIGCOMM 2001*, 2001.

[9] P. Druschel and A. rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. *In Proceedings of HotOS VIII*, May 2001.

[10] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. *In Proceedings of SOSP 2001*, Oct 2001.

[11] S. Saroiu et al. A measurement study of peer-to-peer file sharing systems. *In Proceedings of MMCN 2002*, 2002.

[12] R. Bhagwan, K. Tati, Y. Cheng, S. Savage and G. M. Voelker. Total Recall: System Support for Automated Availability Management. *In Proceedings of NSDI 2004*, 2004.

[13] R. Bhagwan, S. Savage, and G. M. Voelker. Replication Strategies for Highly Available Peer-to-peer Storage Systems. *In Proceedings of FuDiCo*, June 2002.

[14] Z. Xu, R. Min and Y. Hu. Reducing Maintenance Overhead in DHT Based Peer-to-Peer Algorithms. *In Proceedings of P2P 2003*, 2003.

[15] C. Blake and R. Rodrigues. High Availability, Scalable Storage, Dynamic Peer Networks : Pick Two. *In Proceedings of HotOS-IX*, May 2003.

[16] R. Bhagwan, S. Savage, and G. M. Voelker. Understanding Availability. *In Proceedings of IPTPS 03*, 2003.

[17] L. Carcs-Erice, E. W. Biersack, P. Felber, K. W. Ross and G. Urvoy-Keller. Hierarchical Peer-to-Peer Systems. *In Proceedings of Euro-Par 2003*, 2003.

[18] B. Yang and H. Garcia-Molina. Designing a Super-Peer Network. *In Proceedings of ICDE 2003*, 2003.

[19] Bittorent, http://bittorent.com

[20] Gnutella, http://gnutella.wego.com/, 2003.

[21] J. Liang, R. Kumar, and K.W. Ross. Understanding KaZaA. *http://cis.poly.edu/ ross/papers/UnderstandingKaZaA.pdf*, 2004.

[22] E. Cohen and S. Shenker. Replication Strategies in Unstructured Peer-to-Peer Networks. *In Proceedings of SIGCOMM02*, 2002.

[23] G. Utard and A. Vernois. Data Durability in Peer to Peer Storage Systems *In Proceedings of CCGrid 2004*, 2004.

[24] Q. Lv, P. Cao, E. Cohen, K. Li and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. *In Proceedings of ICS 2002*, 2002.

[25] J. Kangasharju, K. W. Ross and D. A. Turner. Optimal Content Replication in P2P Communities. *manuscript*, 2002.

[26] M. J. Freedman and D. Mazieres. Sloppy hashing and self-organizing clusters. *In Proceedings of IPTPS03*, 2003.

[27] Y. Zhu and Y. Hu. Efficient, Proximity-Aware Load Balancing for DHT-Based P2P Systems. *IEEE TRANSACTION ON PARALLEL AND DISTRIBUTED SYSTEMS*, Vol 16, No 4, April 2005.

[28] L. Xiao, Z. Zhuang and Y. Liu. Dynamic Layer Management in Super-peer Architectures. *IEEE TRANSACTION ON PARALLEL ADN DISTRIBUTED SYSTEMS*, Vol 16, No 11, November 2005.

[29] P. Backx, T. Wauters, B. Dhoedt, and P. Demeester. A Comparison of Peer-to-Peer Architectures. *In Proceedings of Eurescom Summit*, 2002.

[30] F.E. Bustamante and Y. Qiao. Friendships that Last: Peer Lifespan and Its Role in P2P Protocols. *In Proceedings of Int'l Workshop Web Content Caching and Distribution*, 2003.

[31] N. Daswani, H. Garcia-Molina, and B. Yang. Open Problems in Data-Sharing Peer-to-Peer Systems. *In Proceedings of Ninth International Conference Database Theory*, 2003.

[32] Z. Ge, D.R. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley. Modeling Peer-Peer File Sharing Systems. *In Proceedings of IEEE INFOCOM*, 2003.

[33] K.P. Gummadi, R.J. Dunn, S. Saroiu, S.D. Gribble, H.M. Levy, and J. Zahorjan. Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. *In Proceedings of 19th ACM SOSP*, Oct. 2003.

[34] K. C. Hsiao and C. T. King. A Tree Model for Structured Peer-to-Peer Protocols. *In Proceedings of Third CCGRID*, May 2003.

[35] R. J. Dunn, J. Zahorjan, S. D. Gribble and H. M. Levy. Presence-Based Availability and P2P Systems. *In Proceedings of P2P 2005*, 2005.

[36] S. Rhea, D. Geels, T. Roscoe and J. Kubiatowicz. Handling Churn in a DHT. *In Proceedings of USENIX Annual Technical Conference*, June 2004.

[37] Z. Xu, M. Mahalingam, and M. Karlsson. Turning Heterogeneity into an Advantage in Overlay Routing. *In Proceedings of IEEE INFOCOM*, Apr. 2003.

[38] Z. Zhang, S. Shi, and J. Zhu. SOMO: Self-Organized Metadata Overlay for Resource Management in P2P DHT. *In Proceedings of Second IPTPS*, Feb. 2003.

[39] D.R. Karger and M. Ruhl. Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems. *In Proceedings of Third IPTPS*, Feb. 2004.

감 사 의 글

이 논문이 나오기까지 많은 분들의 도움이 있었습니다. 모든 분들께 감사드립니다. 먼저 지난 8년이라는 오랜 기간동안 부족한 저에게 많은 격려와 충고로 항상 배려해주신 박대연 교수님께 감사드립니다. 교수님의 열정과 가르침은 저에게 많은 표본이 되었고 항상 용기를 주셨습니다. 또 바쁘신 와중에도 논문 심사를 맡아주시고 부족한 부분을 지적해주신 박규호 교수님, 성단근 교수님, 윤현수 교수님, 이광형 교수님께 깊은 감사를 드립니다.

긴시간을 동고동락해온 시스템 소프웨어 실험실의 선후배들에게도 감사드립니다. 항상 가족같이 서로를 아껴주는 분위기의 실험실에서 함께 공부할수 있었다는 것은 저에게 행운 이었습니다. 프리젠테이션의 마법사 정훈형, 느긋하면서도 예리한 멋진 동기 근태, 여러 일들을 벌려서 실험실을 바쁘게 만들던 용주, 꼼꼼하고 세심한 새신랑 동국, 후배들을 잘 보살펴주는 재섭, 운동도 공부도 열심인 용, 책임감이 강한 상권, 항상 열심인 모범생 규동, 모든일에 적극적인 현빈, 예리한 귀염둥이 상환, 순진한 터프가이 기섭, 착한 새내기 희진, 실험실의 살림꾼 명숙씨에게 감사드립니다. 졸업했지만 항상 지켜봐주시고 도움을 주신 선후배분들에게도 감사드립니다. 실험실의 전설 병학형, 강한 카리스마의 연섭형, 멋진 삼성맨 철호형, 터프한 교수님 우현형, 항상 먼가 잘 되는 상호형, 프로그래밍의 귀재 양우형, 언제나 고마운 재선형, 멋진 동기 용진, 스웨덴에 있는 동은, 드디어 아빠가 된 승원형, 멀리서 공부하고 있는 재웅형, 열심히 일하고 있을 진수형과 상엽형, 항상 열심히하던 우진, 투덜대면서도 부지런한 영배, 독특한 성격의 소유자 병집, 의젓한 응신형, 생각이 깊은 필성형, 생각하는 천하장사 대원, 운동을 좋아하던 달리기맨 근태, 많이 챙겨주지 못한 고등학교 후배 화신, 스타일리쉬한 용재, 외모만 아저씨지 알고 보면 착한 지수, 이 모든 분들게 감사의 말씀을 드립니다.

대전에서의 오랜 생활을 같이 해온 경석, 승은, 계태, 성진등 전남과학고

모든 친구들에게 감사의 말을 전합니다.

지금의 제가 잇기 까지 가장 힘이 되어준 것은 저의 가족이었습니다. 먼저, 맏아들로써 집에서 일찍 떨어져 나와 공부하느라 신경도 잘 못쓰는 저를 항상 믿어주시고 아낌없이 도와주신 사랑하는 부모님, 감사합니다. 내가 집에서 떨어져 있을때 내 몫까지 해준 동생 보준에게 고맙다는 말을 전합니다. 그리고 만날때마다 학자로써의 길에 대해 많은 조언을 주신 장인어른과 항상 따뜻한 미소로 절 맞아주시는 장모님께도 감사드립니다. 그외의 많은 친지분들게도 감사드립니다. 마지막으로, 힘들때나 슬플때나 기쁠때나 즐거울때, 항상 내 옆에서 나를 도와주고 격려해준 사랑하는 하나와 나의 두 귀여운 딸들, 민지, 예지에게 사랑한다는 말과 감사한다는 말을 전합니다.

이 력 서

이 름	:	김 경 백
생년월일	:	1976년 05월 15일
출 생 지	:	Mokpo, Republic of Korea
E-mail주소	:	kbkim@sslab.kaist.ac.kr

학 력

1994. 3. – 1999. 2.	한국과학기술원 학부과정 전기및전자공학과 (B.S.)
1999. 3. – 2001. 2.	한국과학기술원 전자전산학과 전기및전자공학전공 (M.S.)
2001. 3. – 2007. 2.	한국과학기술원 전자전산학과 전기및전자공학전공 (Ph. D)

경 력

1999. 2. – 1999. 11.	Development of Enterprise Web Application Server
1999. 7. – 2000. 6.	Non DEC based Distributed PC Monitor with TxRPC
2000. 3. – 2000. 12.	Development of real-time OS disk file system
2000. 10. – 2001. 9.	Development of the network management system
2001. 4. – 2002. 3.	Development of the web-based distributed system for the network management
2004. 3. – 2004. 12.	Data Synchronization and Transformation for Wireless Mobile Terminals

2005. 3. – 2005. 12. Wearable Ubiquitous Computing

연 구 업 적

1. **Kyungbaek Kim** and Daeyeon Park, *Least Popularity-per-Byte Replacement Algorithm for a Proxy Cache*, In Proceedings of 8th International Conference on Parallel and Distributed Systems (ICPADS 2001), pages 780-788, June 26-29, 2001.
2. Woo Hyun Ahn, **Kyungbaek Kim**, Yong-Jin Choi, and Daeyeon Park, *DFS:A De-fragmented File System*, In Proceedings of 10th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2002), pages 71-80, October 12-16, 2002.
3. **Kyungbaek Kim** and Daeyeon Park, *Subway : Peer-To-Peer Clustering of Clients for Web Proxy*, In Proceedings of The 2003 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2003), pages 1683-1688, June 23-26, 2003.
4. **Kyungbaek Kim**, Woo Jin Kim and Daeyeon Park, *Efficient and Scalable Client-Clustering for Proxy Cache*, In Proceedings of 6th IEEE International Conference on High Speed Networks and Multimedia Communications (HSNMC 2003), pages 83-92, July 23-25, 2003 (LNCS).
5. **Kyungbaek Kim** and Daeyeon Park, *Efficient and Scalable Client Clustering for Web Proxy Cache*, IEICE Transactions on Information and Systems, Vol.E86-D No.9 pp.1577-1585, SEP, 2003.
6. Woo Jin Kim, **Kyungbaek Kim**, Jaesun Han, Keuntae Park and Daeyeon Park, *Thread-Aware Garbage Collection for Server Applications*, In Proceedings of 2004 International Symposium of Applications and the Internet (SAINT 2004), pages 81-87, January 26-30, 2004.

7. **Kyungbaek Kim** and Daeyeon Park, *Caching large files by using p2p based client-cluster for web proxy cache*, In Proceedings of IASTED International Conference on Parallel and Distributed Computing and Networks 2004 (PDCN 2004), page 643-648, February 17-19, 2004.

8. Byoung-Jip Kim, **Kyungbaek Kim** and Daeyeon Park, *The Content-Aware Caching For Cooperative Transcoding Proxies*, In Proceedings of International Conference on Information Networking 2005 (ICOIN 2005), pages 766-775, January 31 - February 2, 2005 (LNCS).

9. **Kyungbaek Kim** and Daeyeon Park, *Efficient Caching Policies for the P2P Web Caching*, In Proceeding of International Conference of Computational Science and its Applications 2005 (ICCSA 2005), May 9-12, 2005

10. **Kyungbaek Kim** and Daeyeon Park, *Efficient Resource Management for the P2P web Caching*, In Proceedings of Service Assurance with Partial and Intermittent Resources Conference (ACIT/SAPIR/ELETE 2005), pp. 382-387, July 17 - 20, 2005.

11. **Kyungbaek Kim** and Daeyeon Park, *Mobile NodeID based P2P Algorithm for the Heterogeneous Network*, In Proceedings of International Conference on Embedded Software and Systems 2005 (ICESS 2005), pp. 485-495, December 16-18, 2005.

12. **Kyungbaek Kim** and Daeyeon Park, *Heterogeneity aware P2P algorithm by using mobile nodeID*, In Proceedings of International Conference on Information Networking 2006 (ICOIN 2006), pp. 975-984, January 16-19, 2006 (LNCS).

13. **Kyungbaek Kim** and Daeyeon Park, *Reducing Data Replication Overhead in DHT Based Peer-to-Peer System*, In Proceedings of International Conference on High Performance Computing and Communications 2006 (HPCC 2006), pp.915-924 , September 13-15, 2006 (LNCS).

14. **Kyungbaek Kim** and Daeyeon Park, *Reducing Outgoing Traffic of Proxy Cache by using Client-Cluster*, Journal Of Communications and Networks, Vol.8 No.3 pp.330-338, SEP, 2006.

15. Hyunbin Lee, YongJoo Song, **Kyungbaek Kim**, Donggook Kim and Daeyeon Park, *CriStore : Dynamic Storage System for Heterogeneous Devices in Offsite Ubiquitous Communities*, In Proceedings of The 22nd Annual ACM Symposium on Applied Computing (SAC 2007), March 11 - 15, 2007.

16. **Kyungbaek Kim**, and Daeyeon Park, *Efficient and tailored resource management for the P2P web caching*, IEICE Transactions on Information and Systems, Accept for Publication. (JAN 2007)