

國立交通大學

資訊科學與工程研究所

碩士論文

在以 DHT(Distributed Hashing Table)為基礎的同儕網路
中對於全域性信任及評比管理的有效演算法

Scalable Algorithms for Global Trust and Reputation
Management in DHT-based Peer-to-Peer Networks

研究生：吳事修

指導教授：邵家健 副教授

中華民國九十五年七月

在以 DHT(Distributed Hashing Table)為基礎的同儕網路中對於全
域性信任及評比管理的有效演算法
Scalable Algorithms for Global Trust and Reputation Management in
DHT-based Peer-to-Peer Networks

研 究 生：吳事修

Student：Shih-Shou Wu

指 導 教 授：邵家健

Advisor：John Kar-kin Zao

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所



Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in
Computer Science

August 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年八月

國立交通大學

研究所碩士班

論文口試委員會審定書

本校 資訊科學與工程 研究所 吳事修 君

所提論文：

Scalable Algorithms for Global Trust and Reputation

Management in DHT-based Peer-to-Peer Networks

合於碩士資格水準、業經本委員會評審認可。

口試委員：張明峰 邵家建
李宜隆 曹孝標

指導教授：邵家建

所長：曾文忠

中華民國九十五年八月十六日

Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
Hsinchu, Taiwan, R.O.C.

As members of the Final Examination Committee, we certify that
we have read the thesis prepared by Shih-Shou Wu
entitled Scalable Algorithms for Global Trust and Reputation
Management in DHT-based Peer-to-Peer Networks

and recommend that it be accepted as fulfilling the thesis
requirement for the Degree of Master of Science.

Committee Members:

<u>Ming-Tsang Chang</u>	<u>Shih-Shou Wu</u>
<u>Guangling Lee</u>	<u>Shih-Li Tsao</u>

Thesis Advisor: Shih-Shou Wu

Director: Wei-Gung Tseng

Date: 2006-8-16

摘要

在這篇論文中，我們首先探討了在以 DHT (Distributed Hashing Table) 為基礎的同儕網路中，關於資源提供者選擇的問題。使用者總是希望能在眾多提供所需資源的端點中選擇最適合的一個以獲得此資源。但同儕網路本身開放及匿名的本質，卻限制了使用者做選擇時的資料依據。因此我們提出了以評比管理及信任關係為主軸的機制，讓使用者能獲取他人的意見並對這些意見進行整合及評估，最後做出適當的選擇。在這樣的機制中，我們調整了關於評比資料的儲存與收集方式，讓使用者能以更快的收集到更多的資訊。我們也評估收集到資訊的可信度，並以資訊本身的可信度來決定此資訊在決定過程中所佔的比重，希望能提高使用者做決定的準確性。接著我們對於演算法中收集評比資料的數量做了驗證，證明我們所提出的機制的確能達到我們的目標。最後，我們提出研究的結論以及未來值得研究的問題。



Abstract

In this thesis, we discuss the problem of the selection of resource providers in the DHT-based peer-to-peer networks. Users always hope to choose the most appropriate provider to get the wanted resources from lots of peers providing the resources. However, the dynamic and anonymous nature of the networks limits the information users can get. So we present a mechanism based on reputation management and trust relations. In such mechanism, users can get the opinions from other peers, and they aggregate and evaluate these opinions to make proper decisions. In our works, we adjust the storage and gathering of reputation scores to let users get more information by less cost. We also estimate the trustworthiness of the reputations collected and weight these reputation scores by their trustworthiness to improve the precision of the decisions of users. Then we verify the amount of reputations users can get in our algorithm to prove that our mechanism achieves the goal. Finally, we describe the conclusion and future works of our research.

致謝

首先我要感謝邵家健教授。在這兩年中，他帶領我進入這塊領域，並教導我如何對問題進行思考、研究，讓我慢慢的成長。在研究的最後階段，他也不吝惜的對問題提出指導與意見，導正研究的方向。今天能有這樣的結果，還是要歸功於他。老師謝謝你。

另外還要感謝曾經幫助過我的李官陵教授。感謝她親切的笑容與招待，讓我即使身處人生地不熟的地方，仍有如沐春風的感覺。

在我沮喪與無助的時候，身邊總有一些好朋友能夠適時的給我支持，他們總是相信我能夠撐到最後，鼓勵我往前進。我真的感謝這些朋友，他們陪我走過最後一段黑暗的日子，我只能說，你們永遠是我的好朋友，謝謝。

最後，我必須感謝我的家人，他們對我的付出與關心，我永遠也說不完。之前研究陷入困境，他們比我還擔心；而這篇論文最後能完成，他們比我還開心。我只想說：我做到了，不會讓你們失望的。

謹於此感謝所有幫助我的人以及我最親愛的家人。

Contents

摘要.....	i
Abstract.....	ii
致謝.....	iii
Contents	iv
List of Figures	vi
List of Tables.....	vii
Chapter1 Research Overview	1
1.1 Problem Statement	1
1.2 Project Approach.....	2
1.3 Outline of Thesis.....	3
Chapter2 Background Knowledge.....	4
2.1 Peer-to-Peer Networks	4
2.1.1 The Unstructured Peer-to-Peer Networks	4
2.1.2 The Structured Peer-to-Peer Networks	6
2.1.3 Chord.....	7
2.2 Trust-based Reputation Management	9
2.2.1 Reputation System	9
2.2.2 Trust relation and Global Trust.....	10
2.3 Related Work.....	12
2.3.1 EigenTrust.....	12
Chapter 3 Concepts of Research.....	15
3.1 Principles and Objective	15
3.2 Maintenance of Reputation Scores	16
3.2.1 Storage of Reputation Scores.....	17
3.2.1.1 Search Tree of Chord	17
3.2.1.2 Recursive Storage of Reputation Scores.....	19
3.2.1.3 Complementary Nodes.....	20
3.2.1.4 Determination of Complementary Nodes	21
3.2.1.5 Usage of Complementary Nodes	23
3.2.2 Handling of Received Reputation Scores	24
3.2.2.1 Aggregation of Reputation Scores	25
3.2.2.2 Improvement of Aggregation Function.....	25
3.2.3 Gathering of Reputation Scores	26
3.2.3.1 Search Agents.....	26
3.2.3.2 Collecting Reputation Scores.....	27

3.2.3.3 Use Complementary Nodes in Reputation Collection	29
3.3 Verification of Collected Information	30
3.3.1 Properties of Chord Search tree	30
3.3.2 Verification Scheme	31
3.4 Other Improvement of The Mechanism.....	33
3.4.1 Temporal Change	33
Chapter 4 Research Result	35
4.1 The Influences of Skip Distance	35
4.2 The Relations between Search Trees and Pascal Triangle	36
4.3 The Number of Nodes Which An Aggregated Score Represents	37
4.4 The Information Can Be Collected in the Reputation Collection.....	39
Chapter 5 Conclusion and Future Works	44
5.1 Conclusions.....	44
5.2 Future Works.....	44
Reference	46



List of Figures

Fig.1	The architecture of unstructured networks	5
Fig.2	Large number of packages caused by request flooding	5
Fig.3	An example of successor peers in the Chord ring	7
Fig.4	The finger tables stored in each node in the Chord network	8
Fig.5	An example of trust transitivity	11
Fig.6	An example of parallel trust combination	12
Fig.7	Overview of the whole mechanism	16
Fig.8	The search tree of 32-node Chord network	18
Fig.9	The search tree of 32-node Chord network with $n = 12$	19
Fig.10	The pairs of complementary nodes and their searching paths	21
Fig.11	One of the pairs of complementary nodes	22
Fig.12	All the pairs in the searching tree	23
Fig.13	The demonstration of the storage of the reputation of <i>peer</i> (15) about <i>peer</i> (12)	24
Fig.14	The scheme of recursive storage of reputation scores with $s = 2$	28
Fig.15	Requests to the complementary node for additional information	29
Fig.16	The recursive structure of the search tree in Chord	31
Fig.17	The nodes in the search paths of <i>peer</i> (15) and <i>peer</i> (26)	32
Fig.18	The aggregation of the scores in different time interval	34
Fig.19	Pascal triangle	36
Fig.20	The number of nodes in each layer of the search tree	37
Fig.21	The ratio of the reputations stored in the root of a tree to the total number of nodes in the tree	38
Fig.22	The search tree of a 32-node Chord network	40
Fig.23	The ratio of the collected information along the longest path to the total number of nodes in the network	42

List of Tables

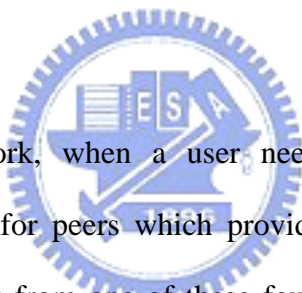
Table.1	The number of nodes in each layer of the search tree	37
Table.2	The result of computation under different values of m and s	38
Table.3	The ratio of the collected information along the longest path to the total number of nodes in the network	41



Chapter1 Research Overview

In the recent years, the peer-to-peer networks are widely applied in many services, and the number of data shared through peer-to-peer networks increases continuously. Besides the convenience of services, users pay more and more attention to the security and quality of resources they receive. In this chapter, we will give a problem statement. Then, we introduce the outline of this thesis after a brief description of the approaches we use.

1.1 Problem Statement



In a peer-to-peer network, when a user needs some resources, the most convenient way is to search for peers which provide the wanted resources in the network and get the resources from one of these found peers. The main problem is how to select the best provider of wanted resources in a DHT-based peer-to-peer network [1].

Because there are no central servers in the DHT-based peer-to-peer networks and peers always join and leave dynamically, it is difficult for users to have information of all the peers. Without enough information of the found peers, users have to ask for other peers' recommendations to make appropriate decisions. But in a DHT-based environment, because each user maintains information of less other peers and the length of the searching path of each peer is limited, the number of peers which each peer may know or contact is obviously also restricted. Thus the first question to be solved is how to let users collect recommendations as many and efficient as possible.

After gathering recommendations from other peers, the next work is to combine and evaluate the reputations contained in these recommendations. By aggregating these reputation scores, users can realize the previous experiences of other peers and make their own decisions more precisely. When handling these recommendations from different peers, it is necessary to take the trustworthiness of the peers providing the recommendations into account. The trustworthiness of a peer is determined by the correctness of the previous recommendations it provided. Peers may trust the same peer in different degrees. So the second question to be solved is how to derive a fair and precise result from the collected recommendations.

1.2 Project Approach

In this thesis, we develop a decentralized mechanism to solve the problem mentioned above in DHT-based peer-to-peer networks. To resolve the first question, we let each user record and update the reputations assigned to other peers after interacting with them, such as receiving resources from them. To save the time for collecting the recommendations, peers store and gather the reputation scores stored along their own searching paths. And we also apply some scheme to improve the total algorithm.

To solve the second question, we propose an aggregation scheme to aggregate the reputations received. The reputation scores will be weighted by the number of nodes which they represent and then combined to compute a value which can really reflect the experiences of other peers in the network with one certain peer. Moreover, the weight of each score will be adjusted any time according to the difference between it and other scores or the average values. So the false reputation scores will have less weight than the true reputations and thus have less influence on the derived values.

1.3 Outline of Thesis

The remaining of this thesis is organized as follows. Chapter 2 describes the background techniques we use in our research and the related work of trust management, EigenTrust. In chapter 3, we introduce the algorithm and mechanism of our works. And chapter 4 verifies the results of our works. Finally, chapter 5 is the conclusion and future works.



Chapter2 Background Knowledge

In this chapter, we will shortly introduce the background technologies we use in our work, including peer-to-peer networks in section 2.1 and trust-based reputation management in section 2.2. Then we will introduce EigenTrust, the related work of our research, briefly in section 2.3.

2.1 Peer-to-Peer Networks

In the beginning, data communication on the computer networks is under client-server architecture. Clients communicate with servers by authenticated protocols and ports. And servers have to take responsibility for managing the resources and the power of the other peer nodes. Over the Internet today, the network environments are more complex. For the convenience of data exchange, peer-to-peer architecture is invented. Instead of building new physical networks, peer-to-peer networks just modify the communication paths between users over the original physical networks. Contrary to the client-server architecture, there are no central servers in the peer-to-peer networks. All data communication is completed through the direct or indirect transmission between peer nodes in the network.

We will describe the unstructured and the structured peer-to-peer networks in the following sections.

2.1.1 The Unstructured Peer-to-Peer Networks

The architecture of the unstructured peer-to-peer networks is based on the concept of neighbor nodes. Every peer node will find its neighbors when joining the

network. And when peer nodes need some wanted resources, they will send requests for the resources to all their neighbors. After receiving requests, the neighbor nodes will check the requests and help to forward the requests to their own neighbors. If a peer node receiving requests has the wanted resources, he will return the searching result to the original node so that the original node can get the resources from him. (Fig.1)

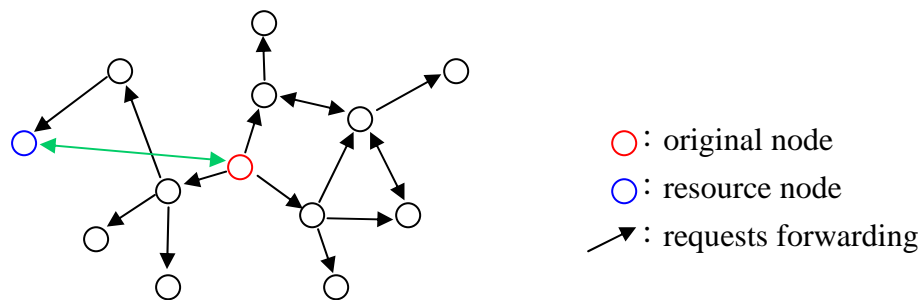


Fig.1: The architecture of unstructured networks

An example of such architectures is Gnutella. Because it is a distributed network environment, the whole system will not easily be paralyzed when some nodes are broken. But in such architecture, we can not make sure of the length of the searching paths. And the number of request packages will be larger and larger as requests are continuously forwarded. (Fig.2)

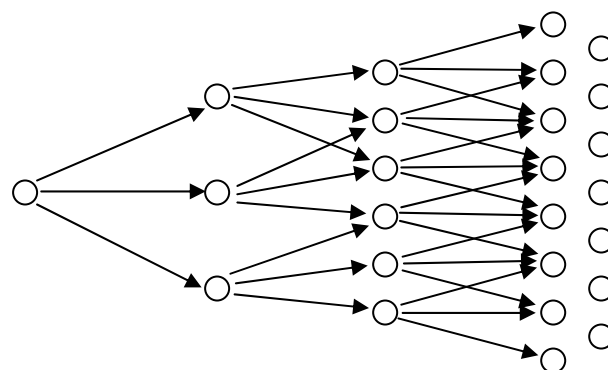


Fig.2: Large number of packages caused by request flooding

2.1.2 The Structured Peer-to-Peer Networks

The structured peer-to-peer networks always use Distributed Hashing Table (DHT) to help to organize the peer nodes in the network. DHT assigns every peer node and resource an identifier by hash functions to reduce the number of unnecessary packages. Instead of using flooding when forwarding request packages, it selects the target of request packages and effectively controls the number of request packages sent and the length of searching time.

When a peer joins the network, its identifier is chosen by hashing some of the peer's information, such as IP address, port, etc. According to the identifier, the peer determines its neighbor nodes which may receive the requests it sends. Additionally, identifiers of resources are produced by hashing some attributes of the resources, such as name, size, etc. The identifiers of resources will be mapped to the peer nodes existing in the network and the meta data of the resources will be stored in the mapped nodes. When searching resources, users have to use the identifiers to find the meta data and get the actual location of the resources.

To search or communicate with a peer node with given identifier, users have to select the node which is the closest to the target node from all their neighbor nodes and send requests to it. Then this selected neighbor node will help to forward the requests by the same way. Instead of broadcasting the requests to all neighbors, every node in the searching path only sends requests to one of its neighbor nodes. Thus this mechanism solves the problem of package flooding in the second generation of peer-to-peer networks, and it also effectively reduces the length of searching path.

There are many implementations of the structured peer-to-peer networks. Next we introduce one of the DHT-based peer-to-peer networks, Chord.

2.1.3 Chord

Chord [2] is one method of DHT-based peer-to-peer networks. It assigns every peer node and data key an m -bit identifier by hash functions. A peer's identifier is chosen by hashing the peer's IP address, while a key identifier is produced by hashing the data key. Peer nodes are ordered on an identifier circle with size of 2^m . The identifier circle is termed as Chord ring. To store the data keys to the mapped peers according to their identifiers, Chord uses the mechanism of "successor". For a key with identifier n , its successor peer is the first peer clockwise from n in the Chord ring, and the successor peer will be denoted by $successor(n)$ in the following description. For example, in a Chord network with $m = 3$, there exist three peers: $node(0)$, $node(1)$, and $node(3)$. Fig.3 shows how to store three data keys: $key(1)$, $key(2)$, and $key(6)$ to the network. The successor of identifier 1 is $node(1)$, so $key(1)$ is stored in $node(1)$. Similarly, the successor of identifier 2 is $node(3)$ and the successor of identifier 6 is $node(0)$. So $key(2)$ would be located at $node(3)$, and $key(6)$ at $node(0)$.

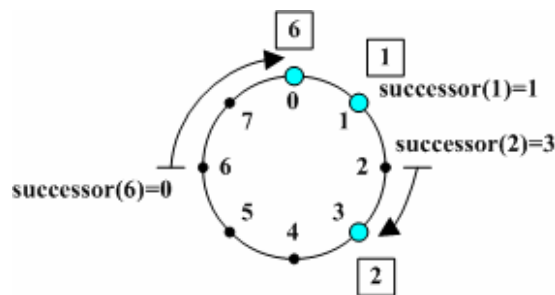


Fig.3: An example of successor peers in the Chord ring

As m is the number of bits in the identifier space, each peer n maintains a routing table with up to m entries, called the finger table. The i^{th} entry in the table at peer n contains the information of the peer $successor(n + 2^{i-1})$, where $1 \leq i \leq m$. Fig.4 shows a Chord network with $m = 3$, and there exists three peers whose identifiers are 0, 1 and 3 respectively. The finger table at every peer n records three entries: the successor

peers of $(n+1)$, $(n+2)$, and $(n+4)$. For example, the finger table at $node(0)$ will record the data of $successor(1)$, $successor(2)$, and $successor(4)$, which are $node(1)$, $node(3)$, $node(0)$. Finger tables have two properties: the first, peers store information about only a small number of other peers in the finger tables, and know more about peers closely following them on the Chord ring than other peers. Second, the finger table does not contain enough information to directly determine the successor of an arbitrary key. For example, $node(3)$ cannot determine $successor(1)$, as $successor(1)$ ($node(1)$) is not present in the finger table at $node(3)$.

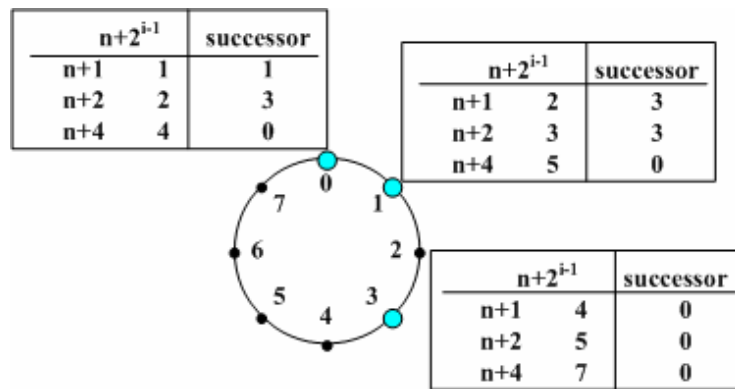


Fig.4: The finger tables stored in each node in the Chord network

When a node n does not know the successor of a key k , it can find a node whose identifier is closer than its own to k . Because of the first property of finger tables mentioned above, that node will know more about the Chord ring in the region of k than n does. Thus n searches its finger table for the node j whose identifier most immediately precedes k , and asks j for the node it knows whose identifier is closer to k . by repeating this process, n learns about nodes with identifiers closer and closer to k . Taking Fig.4 as example, $node(3)$ wants to find $successor(1)$. Since 1 belongs to the circular interval $[7, 3)$, $node(3)$ checks the $successor(7)$ in the finger table, which is $node(0)$. Because 0 precedes 1, $node(3)$ will ask $node(0)$ to find $successor(1)$. In turn, $node(0)$ will infer from its finger table that $successor(1)$ is $node(1)$ itself, and return $node(1)$ to $node(3)$.

By the mechanism of finger tables, the number of nodes that must be contacted to find a successor in an N-node Chord network is $O(\log_2 N)$.

2.2 Trust-based Reputation Management

In many network services, users face each other with a form of pseudonym, such as ID, nickname, etc. The real status of a user is often hidden behind his pseudonym. Under this condition, some users start to cheat or attack the normal users maliciously. For example, in the network auction, both seller and buyer don't know each other before. Without perfect protection schemes, there will be always disputes happened if someone want to cheat other users. But if users can know the credit or performance of the other side in advance and then select their targets of transactions, the rate of successful transactions will be improved. To do this, we can use reputation systems to record the usual performance of users and use the idea of trust to strengthen the functions of the reputation systems [3]. We will describe the ideas of reputation and trust respectively in the following sections.

2.2.1 Reputation System

Reputation systems [4] let users score and record the performances of other users which they have interacted with. eBay [5] is a simple example. In eBay, both seller and buyer give each other a reputation after the transactions, and the reputation may be positive (+1), negative (-1), or neutral (0). The total score of a seller or a buyer is the sum of all his reputations received. Generally speaking, a user with higher total scores always has more satisfied traditions than unsatisfied ones. Users can select who to trade with by examining the total scores of other users.

Reputation systems record the subjective opinions of users, so we can not make

sure that all users assign the reputations fairly. And that will cause some problems. For example, user A gives other users negative reputations on purpose to reduce the total scores of other users and to get the chances of transactions. In another situation, user A can collude with some other users and get positive reputations from them. So the total score of A increases and thus A can get the chances of transactions.

Additionally, the combination and evaluation of reputation scores are also important. For example, there is a reputation system which only records the sum of positive and negative reputations but doesn't keep track of the real condition of transactions. Malicious users may be honest in the traditions with lower value to accumulate positive reputations and wait for chances to cheat in the traditions with high value. In such reputation system, the total scores can't completely represent the actual performance of a user.

In a word, in a private and dynamic peer-to-peer network, a well-designed reputation system will effectively protect the peer nodes from malicious cheats and attacks [6][7].



2.2.2 Trust relation and Global Trust

Because of peer-to-peer networks and E-Commerce, the idea of trust is applied in the field of computer networks gradually [8]. Trust relations are composed of five main components: trust origin, trust purpose, trust target, measures, and time. The first three components describe the main concept of trust relations, and measures represent the magnitude of trust relations. There are several forms to express the measures of trust relations, such as binary (trust or not), discrete (strong trust, weak trust, weak distrust, strong distrust), continuous (probability), etc. Moreover, the content or the strength of trust relations always change as time goes by. The time component of trust

relations is used to distinguish the trust relations at different time slots.

Trust relations originally indicate that one peer trusts the other peer. Trust transitivity and parallel trust combination are the extension of the trust relations for usage among more than two peers. Transitivity of trust relations always occurs with recommendation. For example, C has helped B to repair his car, so B trusts C to be a good car mechanic. This is the trust relation between B and C. Now A needs a car mechanic and asks B who can help him. Based on B's own experiences and the trust in C, B commends C to A. And A also trusts B's recommendation. Fig.5 shows the condition:



Fig.5: An example of trust transitivity

In the trust relation in Fig.5, we define the trust relation between B and C as direct trust, and the trust relation between A and B is defined as indirect trust. The direct trust is based on the experience while the indirect trust is the trust in others' recommendations. Trust transitivity is composed of a chain of indirect and direct trust. But actually, trust is not implicitly transitive, because trust is weakened or diluted through transitivity. Take Fig.5 as example, the trust between A and C will not be stronger than the one between A and B or between B and C. The trustworthiness will be lower and lower after several hops of recommendation.

While trust transitivity is the series connection of trust relations, parallel trust combination is the parallel connection of trust relations. It is caused by several commendations with the same target (as Fig.6). Unlike the weakened trust relations caused by trust transitivity, the parallel combination of positive (or negative) trust relations has the effect of strengthening the derived trust.

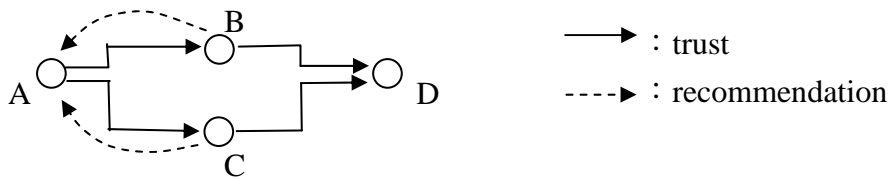


Fig.6: An example of parallel trust combination

By trust transitivity and parallel trust combination, trust relations are expanded and applied in the peer-to-peer networks. In the peer-to-peer networks, every peer node assigns its trust values in other peers. Because trust relations are also subjective judgment, every peer always has different trust value in the same target. To consider the trustworthiness of a peer node, someone presented the idea of global trust. Compared to the local trust values assigned by peers, global trust is to integrate the local trust values of the peer nodes in the network and compute a value to reflect the experiences of all peers in the network with one certain peer. We can estimate the real trustworthiness of one peer by examining its global trust value. There have been many great researches on trust management in the peer-to-peer networks [9], and EigenTrust is one of the works about global trust. We will describe it in Section 2.3 later.

2.3 Related Work

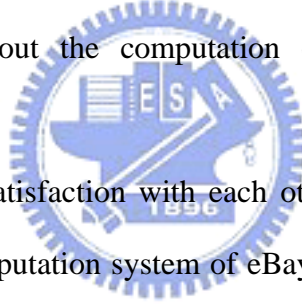
In this section, we will describe one of the related works of the background knowledge mentioned above, EigenTrust [10]. EigenTrust focuses on the trust management and implements the idea of global trust.

2.3.1 EigenTrust

Peer-to-peer file sharing networks have many benefits over standard client-server

approaches to data distribution, including convenience and efficiency. However, the open and anonymous nature of these networks leads to a lack of accountability for the content a peer puts on the network and opens the door to abuses of these networks by malicious peers. Therefore, it is important to ensure that peers obtain reliable information on the quality of resources they are receiving.

In the concept of EigenTrust, attempting to identify malicious peers that provide inauthentic files is superior to attempting to identify these files themselves because malicious users can easily generate a virtually unlimited number of inauthentic files if they are not banned from joining the network. EigenTrust assigns each peer a unique global trust value that reflects the experiences of all peers in the network with it. According to the global trust values, users can identify malicious peers more easily. Following description is about the computation of global trust values in the EigenTrust.



First, peers record the satisfaction with each other after exchanging files. The simplest method is like the reputation system of eBay: each peer i stores the number of satisfactory transactions it has had with peer j , $sat(i, j)$ and the number of unsatisfactory transactions it has had with peer j , $unsat(i, j)$. Then we define s_{ij} as the local trust value of peer i with peer j :

$$s_{ij} = sat(i, j) - unsat(i, j) \quad (1)$$

Having the local trust values between peers, we can compute the global trust values by aggregating the local trust values assigned to one peer by other peers. To aggregate local trust values, it is necessary to normalize them. So c_{ij} is defined as normalized local trust value:

$$c_{ij} = \frac{\max(s_{ij}, 0)}{\sum_j \max(s_{ij}, 0)} \quad (2)$$

In EigenTrust, the way to aggregate the normalized local trust values is for peer i

to ask its acquaintances about their opinions about other peers and weight these opinions by the local trust values peer i places in them:

$$t_{ik} = \sum_j c_{ij} c_{jk} \quad (3)$$

where t^{ik} represents the trust that peer i places in peer k based on asking his friends.

We can write the formula (3) in matrix notation: we define C to be the matrix $[c_{ij}]$ and \bar{t}_i to be vector containing the values t_{ik} , then:

$$\begin{bmatrix} t_{i1} \\ \vdots \\ t_{ik} \\ \vdots \\ t_{in} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{21} & \cdots & \cdots & c_{n1} \\ \vdots & \ddots & & & \\ c_{1k} & c_{2k} & \ddots & & c_{nk} \\ \vdots & & & \ddots & \\ c_{1n} & c_{2n} & \cdots & \cdots & c_{nn} \end{bmatrix} \begin{bmatrix} c_{i1} \\ c_{i2} \\ \vdots \\ \vdots \\ c_{in} \end{bmatrix}, \text{ that is } \bar{t}_i = C^T \bar{c}_i \quad (4)$$

The trust values derived from formula (4) still reflect only the experiences of peer i and his acquaintances. To get a wider view, peer i may ask his friends' friends and get $\bar{t}_i = (C^T)^2 \bar{c}_i$. Continuing in this manner, peer i will get $\bar{t}_i = (C^T)^n \bar{c}_i$ and have a complete view of the network when n is large enough. With the increasing of n , the trust vector \bar{t}_i will converge to the same vector for every peer i . Namely, it will converge to the left principal eigenvector of C . In other words, \bar{t} is a global trust vector in this model. Its elements, t_j , quantify how much trust the system places peer j as a whole.

Chapter 3 Concepts of Research

3.1 Principles and Objective

The design of the algorithm in our research is mainly based on the following objectives and principles:

1. Users record their opinions about the performance of other users as the local reputation scores ordinarily. For the convenience of reputation collection, we let users additionally store their local reputation scores to other peers in the network.
2. Users can get the information of the resource providers when they search for the wanted resources. To do this, we appoint some peers to be the search agents to collect the reputations when they are free.
3. The lengths of the search paths of the nodes in DHT-based peer-to-peer networks are different. To average the unbalanced condition, we design the scheme of “complementary nodes” in the storage and collection of reputation scores.
4. Because the environment is dynamic, we have to make sure of the time validity of the information users can get.

Fig.7 shows the overview of our approaches.

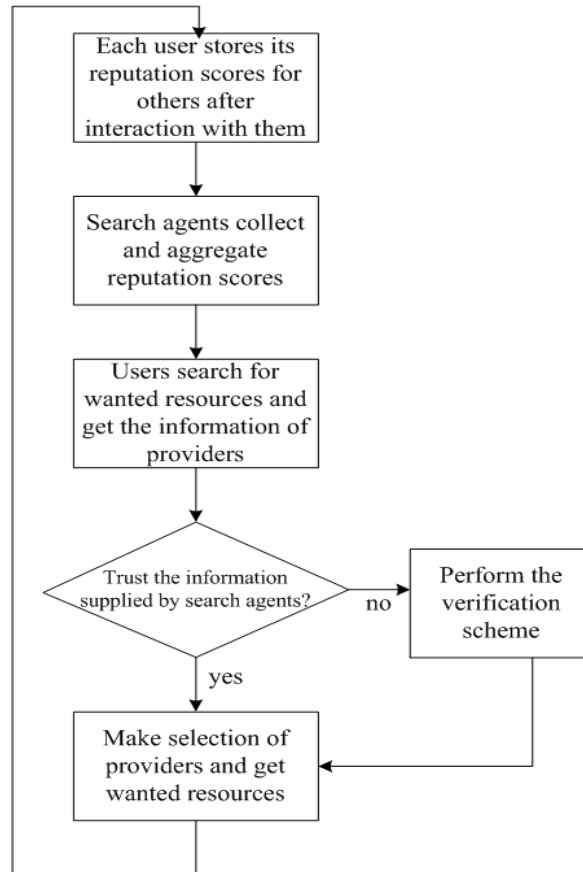


Fig.7: Overview of the whole mechanism

Therefore, we will introduce our algorithm in the following sections. In section 3.2, we describe the maintenance of reputation scores, including the storage and the gathering. Then we will describe the verification of collected information in section 3.3 and discuss some other improvements we do in section 3.4.

3.2 Maintenance of Reputation Scores

From the principles listed in section 3.1, we can know that the maintenance of reputation scores is important to the convenience of reputation collection. How to get the most information by lowest cost is the question which we want to solve. Thus we divide it into two parts and then solve them:

1. To reduce the cost of gathering reputations, we let users collect reputations along their own search paths for the target.
2. To make the whole mechanism scalable, we adjust the storage and gathering of reputation scores.

Next, we will describe the storage and gathering of reputation scores respectively.

3.2.1 Storage of Reputation Scores

Originally, every peer stores the reputation scores it assigns to other peers in itself. And in a DHT-based peer-to-peer network, the length of the search paths from peers to a certain destination peer would be equal to or less than $\lceil \log_2 N \rceil$, where N represents the total number of peers in the network. When a user wants to collect reputation scores along its own search path, it will contact at most $\lceil \log_2 N \rceil$ other peers and thus collect at most $\lceil \log_2 N \rceil$ reputation scores. When N becomes larger, the value of $\lceil \log_2 N \rceil$ will be much smaller than the one of N . To make the mechanism scalable, we adjust the arrangement of the storage of reputation scores and let peers additionally store their reputations in other peers along the search path to improve the number of reputation scores which users can collect along their search path.

Our approach is applicable to DHT-based peer-to-peer architecture in general, so we will take Chord as example to introduce the approaches of our work in the following sections.

3.2.1.1 Search Tree of Chord

In Chord, if we aggregate the search paths for a certain peer of all other peers, we

can build a tree whose root node is the destination peer of these search path. The figure below (Fig.8) shows the search tree of a 32-node Chord network. The root node of this search tree is $peer(n)$, where n is a variable ($n = 0, 1, 2, \dots, 31$).

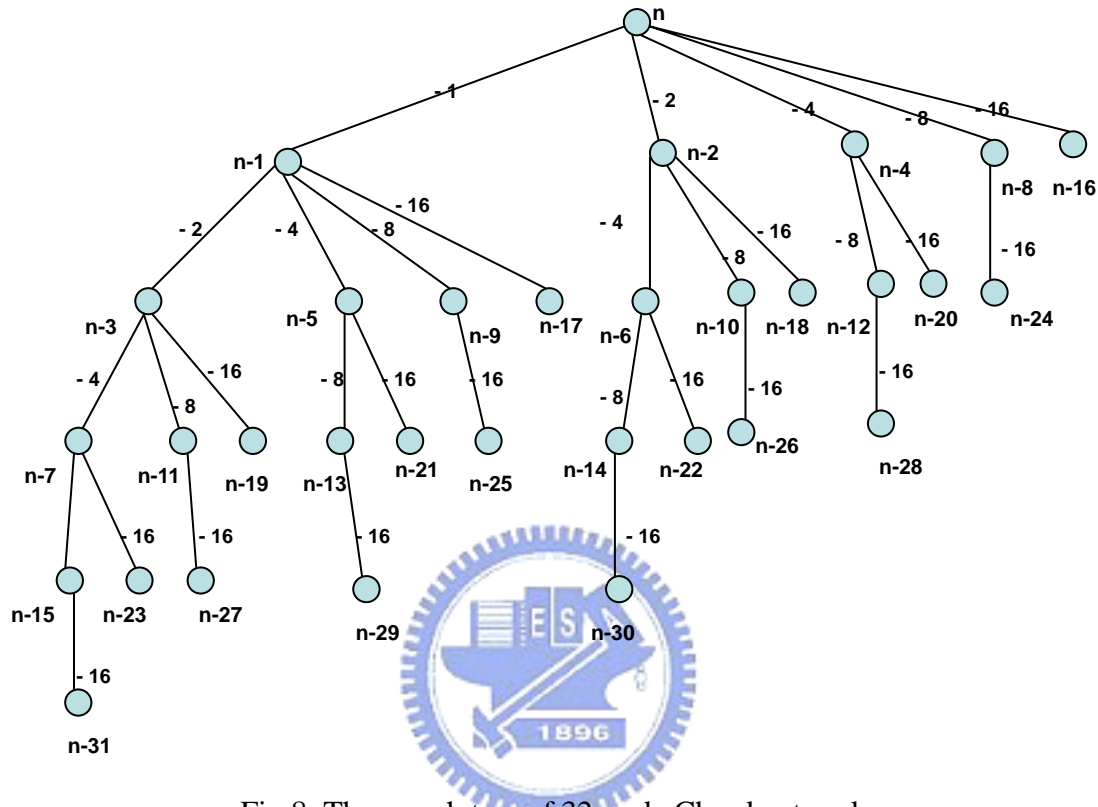


Fig.8: The search tree of 32-node Chord network

These numbers marked in the search tree in Fig.8, such as $(n-27)$, $(n-12)$, etc, actually represent the value of these number modulo 32. The notation of $(\text{mod}32)$ is omitted to be convenient to be showed in the figure. For example, suppose $n=12$, then $(n-4)$ represents 8, and $(n-29)$ represents 15. Fig.9 shows the search tree with $n=12$:

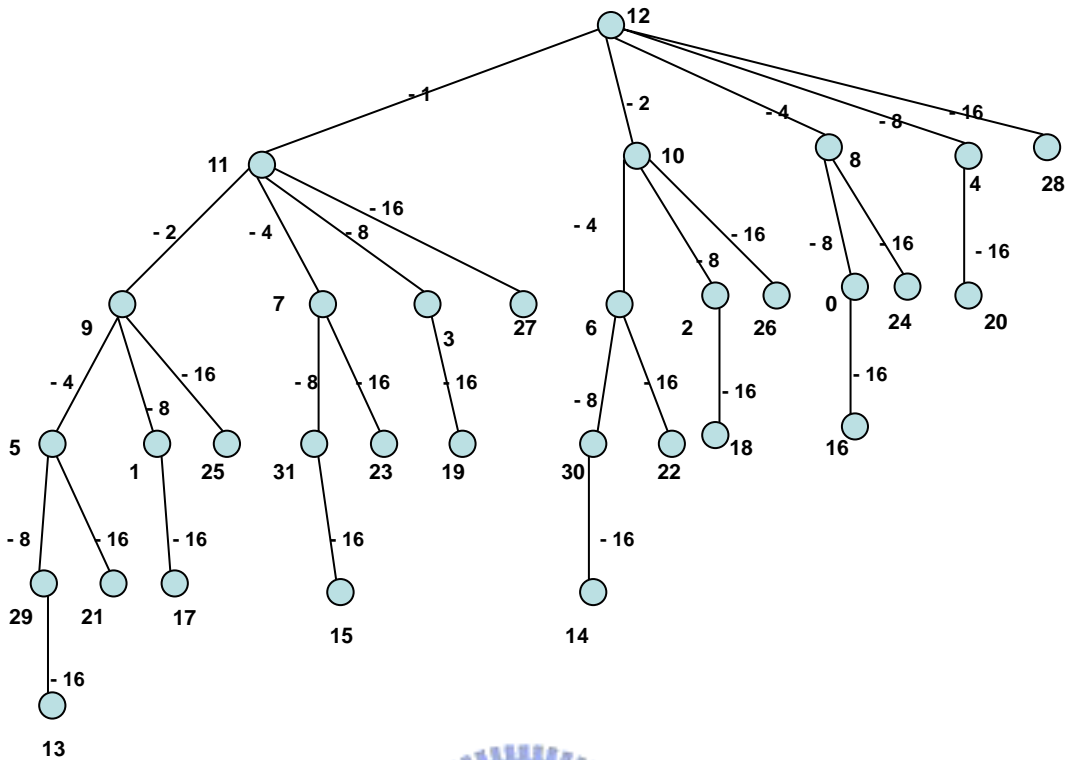


Fig.9: The search tree of 32-node Chord network with $n = 12$

3.2.1.2 Recursive Storage of Reputation Scores

From Fig.9, we can find that the structure of the search tree in Chord has a recursive property. The tree structure can be divided into two subtrees with equal size by the edge marked (-1). We can also say that the search tree of 32-node Chord network is generated by combining two 16-node search trees with an edge. Similarly, we can also generate a 64-node search tree by combining two 32-node search trees. Therefore, to make the scheme of reputation collection scalable, the scheme of reputation storage has to adapt to the recursive property.

In the recursive storage of reputation scores, each peer uploads its own reputation score about one certain provider to the upper nodes along its search path for the provider. We design a variable called “skip distance” which determines which nodes a peer has to upload its reputation. We will use s to denote skip distance in the

following description. For a provider, each peer uploads its reputation about this provider to the nodes which are $s, 2s, \dots$ layers higher than it along the search path for the provider. For example, $s=1$ means that each peer uploads its reputation to all the nodes along its search path for the provider. The influence of s will be described in Chapter 4 later.

3.2.1.3 Complementary Nodes

Each peer uploads its reputations to the upper nodes along its search path. So the nodes with shorter search paths will upload reputations to fewer nodes. That means that there will be fewer copies of the reputations assigned by the nodes with shorter search paths in the networks. To average the unbalance condition, we design the scheme of “complementary nodes”.

In the searching tree of Chord networks, the length of the longest searching path is $\lceil \log_2 N \rceil$ where N is the total number of peers in the network while the length of the shortest one is 1. However, we still can find something interesting in such unbalanced tree structure. In the searching tree in Fig.10, we can divide the searching tree into two subtrees with the same size by the red dotted line. And we can find that: if we connect the searching paths of the two leaf nodes which are symmetrical to the dotted line, the number of internal nodes between the two nodes along the connected path will be $\lceil \log_2 N \rceil$. For example, we connect the searching tree of the first leaf node in the left of the line (*peer(27)*) and the one of the first leaf node in the right of the line (*peer(14)*), which are the two red curves in Fig.10 . Then there are $\lceil \log_2 32 \rceil = 5$ internal nodes between the two nodes: *peer(11)*, *peer(12)*, *peer(10)*, *peer(6)*, and *peer(30)*. Similarly, there are also 5 internal nodes: *peer(1)*, *peer(9)*, *peer(11)*, *peer(12)*, and *peer(8)*, between the sixth nodes in the left and the right of the dotted

line ($peer(17)$ and $peer(24)$), which are the two blue curves in Fig.10.

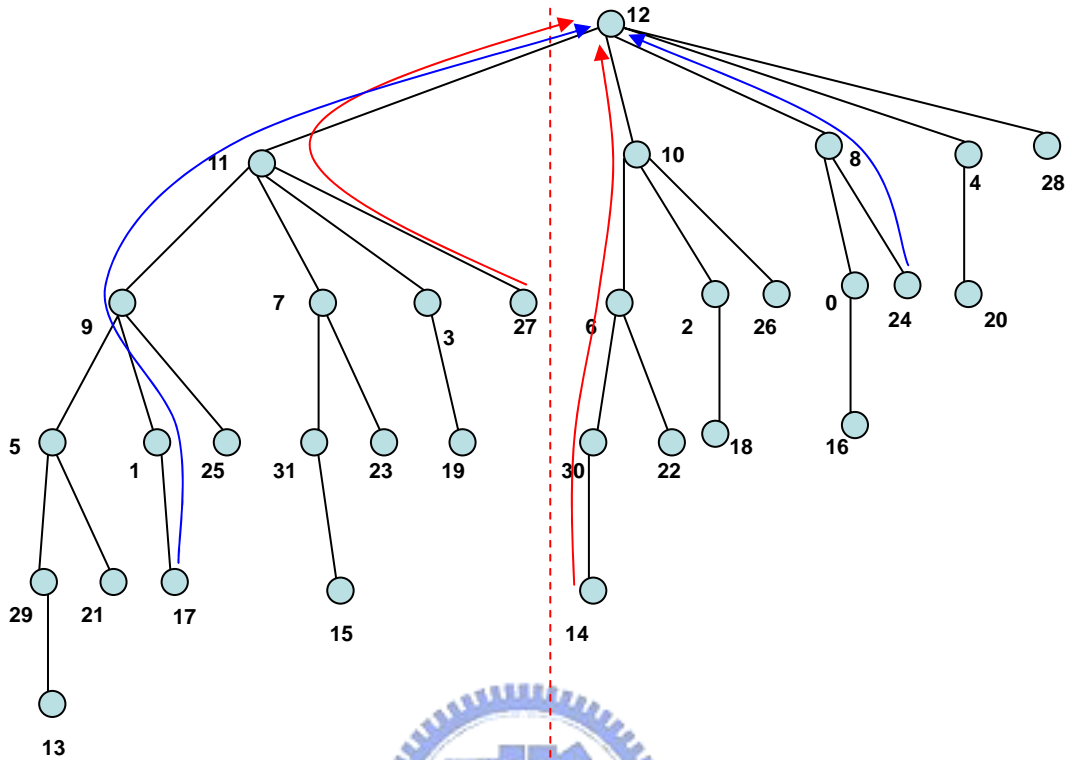


Fig.10: The pairs of complementary nodes and their searching paths.

By the observation above, we can know that: for every leaf node, there exists one another leaf node so that there would be $\lceil \log_2 N \rceil$ internal nodes between the two nodes along their connected searching path. We call that “complementary nodes”. That is, both the nodes are the complementary node of each other.

3.2.1.4 Determination of Complementary Nodes

We introduce the scheme of complementary nodes in the previous section and describe that we divide the searching tree into two parts to determine the complementary nodes. But actually we need a faster way to let a leaf node know which node is its complementary node. And leaf nodes have to determine the complementary nodes by the information they have had instead of determining that

after constructing the whole searching tree.

When two leaf nodes are complementary nodes of each other, the sum of the length of their searching paths must be fixed. And there is also something interesting about the components of their searching paths. Taking the $peer(15)$ and $peer(26)$ in Fig.11 as example, the searching path of $peer(15)$ consists of four edges: (-16), (-8), (-4), and (-1). And the searching path of $peer(26)$ consists of two edges: (-16) and (-2). Totally, the two searching paths have two (-16) edges and one (-8), (-4), (-2) and (-1) edges. There are the same conditions in other pairs of complementary nodes.

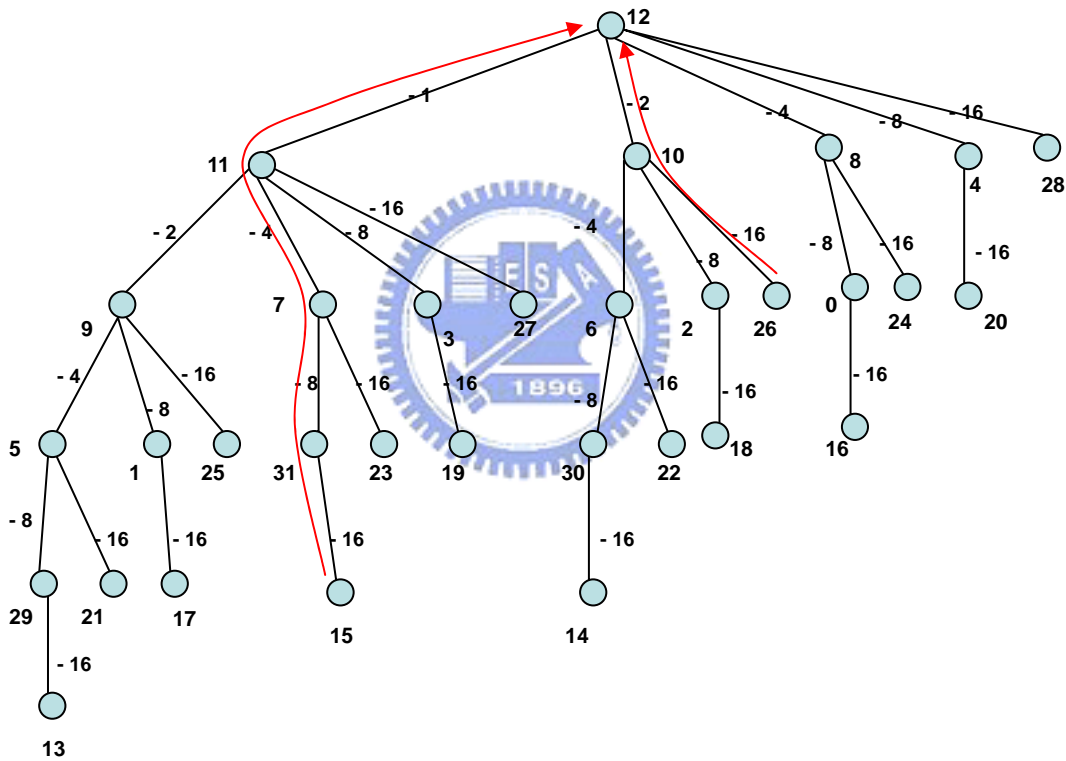


Fig.11: One of the pairs of complementary nodes.

Thus we can conclude that: when two nodes are complementary nodes, the sum of the distance between the two nodes and the root node must be $-(16 * 2 + 8 + 4 + 2 + 1) = -47$. If a leaf node wants to determine its complementary node, it only need to compute the distance from it to the root node first. Then it can know the distance from its complementary node to the root and know the identifier of its complementary node.

For example, $peer(16)$ wants to determine the complementary node when the destination is $peer(12)$. It computes the distance to the root to be (-28) and get the distance from its complementary node to the root to be $(-47) - (-28) = (-19)$. So it can know that its complementary node is $peer(25)$.

Because each internal node in the search tree is connected with a unique leaf node by an edge marked (-16) , we can easily pair all the leaf nodes and the internal nodes (as Fig.12). Therefore, the complementary node of an internal node can be determined by the leaf node which is in the same pair as it.

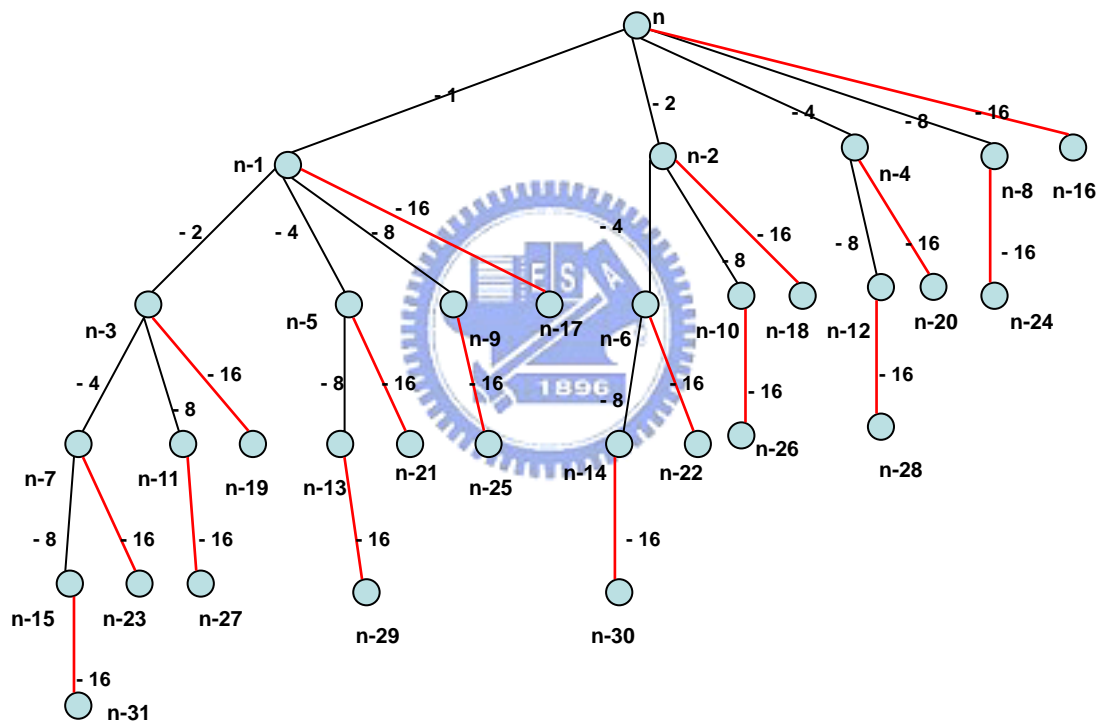


Fig.12: All the pairs in the searching tree.

3.2.1.5 Usage of Complementary Nodes

When a user uploads reputation scores, on one hand it uploads its reputation to the upper nodes along its own searching path, and on the other hand it can send the reputation to its complementary node and ask the complementary node to upload the reputation score to the nodes along the searching path of the complementary node. We

take Fig.11 as example: suppose skip distance $s = 1$, $peer(15)$ uploads the reputation scores about $peer(12)$ to the upper nodes along its own searching path: $peer(31)$, $peer(7)$, $peer(11)$, and $peer(12)$. And it also sends the reputation scores to its complementary node, $peer(26)$. So $peer(26)$ helps to upload the reputation score the nodes along the searching path: $peer(10)$ and $peer(12)$. Totally, there will be 7 nodes storing the reputations about $peer(12)$ of $peer(15)$: $peer(15)$, $peer(31)$, $peer(7)$, $peer(11)$, $peer(12)$, $peer(10)$, and $peer(26)$. Fig.13 shows where $peer(15)$ uploads its reputation scores.

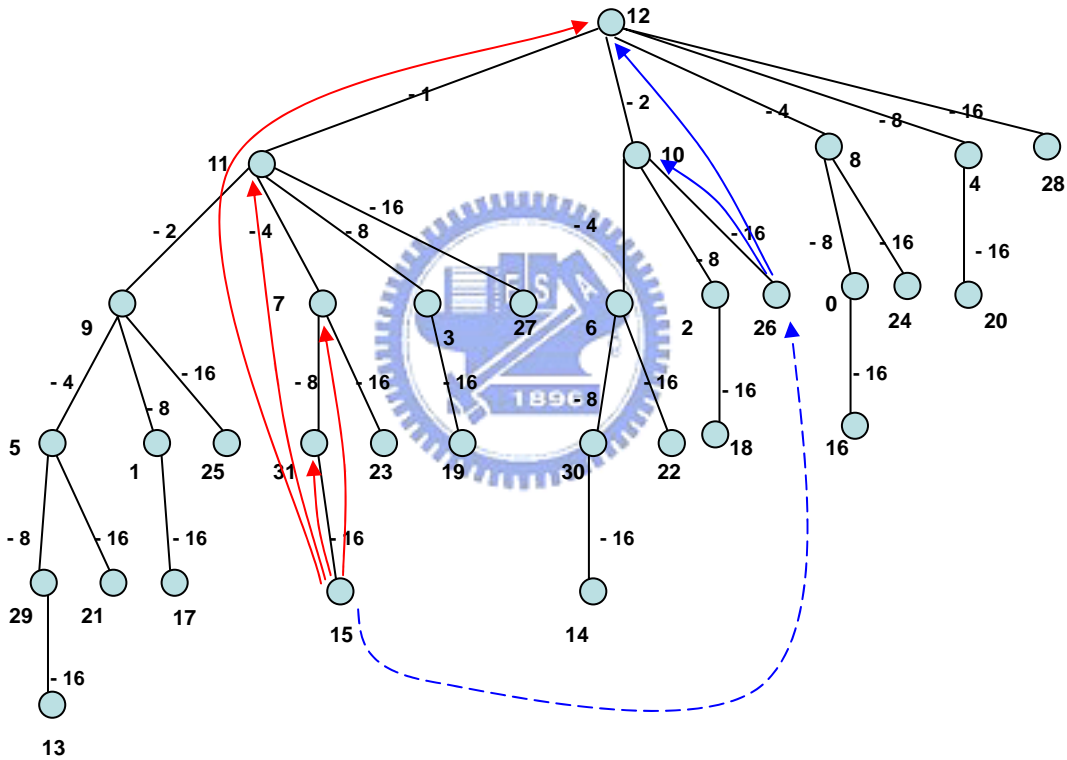


Fig.13: The demonstration of the storage of the reputation of $peer(15)$ about $peer(12)$

3.2.2 Handling of Received Reputation Scores

After receiving the reputations uploaded by the lower nodes in the search tree, how to maintain these reputations is the next question to solve. We can store all the received scores separately. But when the size of the network gets larger and larger, the

amount of reputations a peer stores may be very great. For example, when skip distance $s=1$, the root node of the search tree will store the reputations uploaded by all the lower nodes in the network. The storage of these received reputations will cause a heavy load. So we let peers aggregate received scores and only store the aggregated scores.

3.2.2.1 Aggregation of Reputation Scores

Each peer stores an aggregated score about a provider. When it receives the reputation scores uploaded from other peers, he combines the received scores with the value which he originally stores to generate a new aggregated score.

When combining the received scores and the stored value, the weights of these values depend on the number of nodes represented by these values. For example, suppose there are two scores to be combined. One represents the aggregated scores of four nodes and the other represents the aggregated scores of eight nodes. The more nodes a score represents, the more weight we assign to it. So the weight of the score representing eight nodes will be twice as heavy as the one of the score representing four nodes.

Suppose a peer i stores the aggregated scores of n nodes, \bar{r} . When it receives the reputation score uploaded from another peer j , r_j , it will aggregate the reputation with its stored value and get the new aggregated value, \bar{r}' :

$$\bar{r}' = \frac{\bar{r} * (n - 1) + r_j}{n}$$

3.2.2.2 Improvement of Aggregation Function

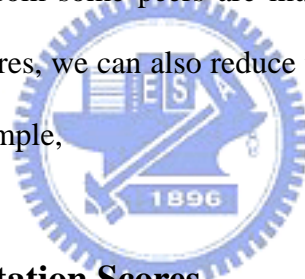
The original aggregation scheme operates aggregation when each score is

received. But if a malicious user continuously uploads reputation score to a peer, it will cause a great computation load to the receiving peer. And the aggregated score stored in this peer will be influenced by the redundantly uploaded scores.

To improve this condition, we set a time interval (ex: 1 hour) for receiving reputations. Peers receive and buffer the reputations received in the same time interval. In the end of this time interval, peers aggregate all the scores buffered with the originally stored value to generate the new aggregated score. For example, in the time interval t , peer i totally receives k reputations: $r_{j1}, r_{j2}, \dots, r_{jk}$. In the end of interval t , peer i aggregates the k reputation scores with the previously stored value \bar{r}_{t-1} :

$$\bar{r}_t = \frac{\bar{r}_{t-1} * (n - k) + r_{j1} + r_{j2} + \dots + r_{jk}}{n}$$

If the scores uploaded from some peers are much different from the scores of other peers or the average scores, we can also reduce the weight of these scores in the aggregation function. For example,



3.2.3 Gathering of Reputation Scores

In DHT-based peer-to-peer networks, users search for the wanted resources to get the information about resource providers. To avoid additional cost, we hope to let users get the information necessary for selection making when they search for the resources. So we design the scheme of “search agent” to perform the collection of reputation scores to reduce the cost of the original users.

3.2.3.1 Search Agents

In the DHT-based networks like Chord, the identifier of the resource is generated by hashing the key of the resource. Then the meta data of the resource will be stored

in the peer which the resource identifier maps to. Because the peer whose identifier is equal to the one of the resource manages the meta data of the resource, we call this peer “resource manager” of the resource. Ordinarily, resource managers own the information of the providers of the resources, such as the IDs and the IP addresses of the resource providers. If a user needs the resources, he has to search for the resource manager to get the information and know which peers provide the resources.

Since all the peers which need the resources have to contact the resource managers, we appoint the resource managers to be the search agents. Search agents are responsible for collecting reputation scores about all the providers of the resources when they are free. After collecting the reputations, search agents combine the gathered scores to generate aggregated values and announce these values in the meta data stored in the resource managers. When users search for the wanted resources and reach the resource manager, they can get the information of the resource providers and also get the aggregated scores of the resource providers. Then users can make selection of resource providers according to these aggregated scores.

3.2.3.2 Collecting Reputation Scores

In our scheme of reputation storage, each peer uploads its reputation scores about the provider to the upper nodes along the search path for the provider. Therefore, when search agents collect reputation scores, we also let them gather reputation scores along their own search paths for the providers.

Because each peer uploads its own reputation to the upper nodes along the search path according to the skip distance s , we can know that if a score is uploaded to a peer n , the score must be also sent to the node which is s layers higher than peer n along the path. We take Fig.14 as example, suppose $s = 2$, then the scores uploaded to

$peer(9)$, $peer(7)$, $peer(3)$, and $peer(27)$ must be also uploaded to $peer(12)$.

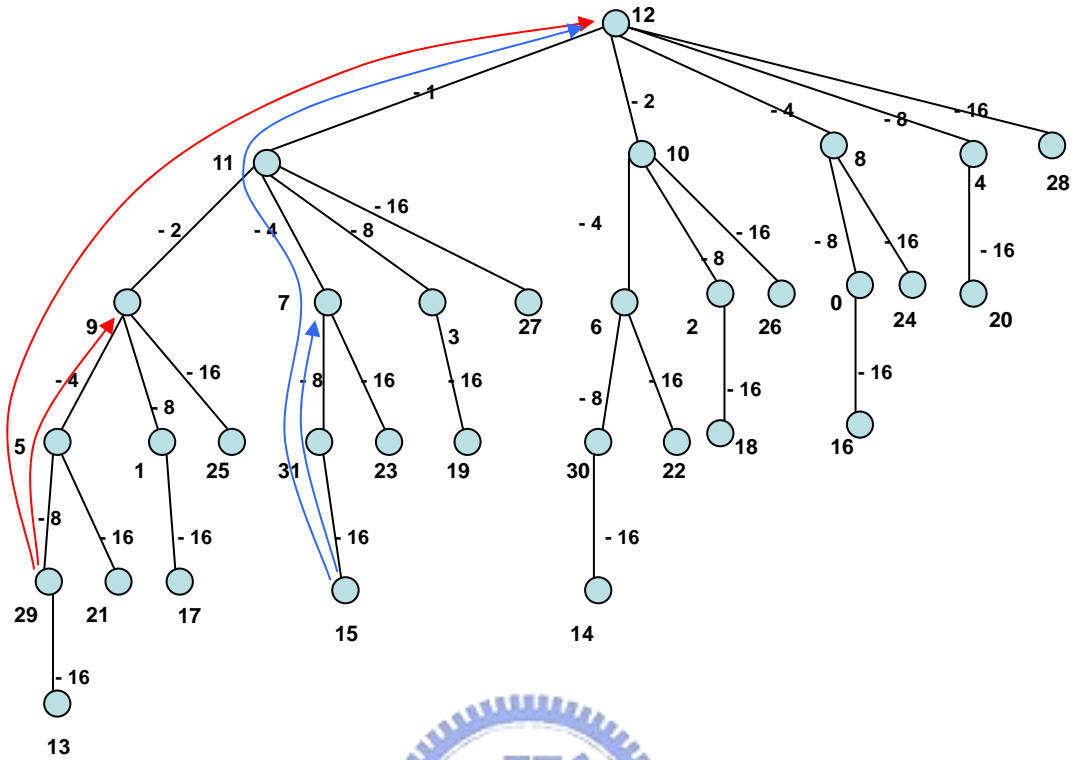


Fig.14: The scheme of recursive storage of reputation scores with $s = 2$

Therefore, we can get all the information contained in a search path by only getting the scores stored in the node in the top s layers of the search tree. Taking the search path from $peer(13)$ to $peer(12)$ as example, the scores uploaded to $peer(9)$ and $peer(29)$ are also uploaded to $peer(12)$ while the scores uploaded to $peer(5)$ and $peer(13)$ are also uploaded to $peer(11)$. So all the information contained in the search path can be got in the $peer(12)$ and $peer(11)$.

The top nodes in the search path store the aggregated scores representing larger amount of lower nodes, so it will cause greater damage if the top nodes are malicious users and supply false aggregated results. So we still collect other aggregated values along the search path to verify the scores supplied by the top nodes in the search path. We will describe the scheme of verification in later sections.

3.2.3.3 Use Complementary Nodes in Reputation Collection

Now, search agents collect reputation scores along their search path for the related providers. But the difference in the length of search path between the nodes in the network will still cause the variance in the amount of information gathered. Similar to the adjustment for reputation storage, we apply the scheme of complementary nodes to the collection of reputation scores.

When a search agent collects reputation scores, he does not only collect the aggregated scores along his search path for the provider, but he also send requests to his complementary node and ask the complementary node to collect reputation scores in the same manner and return the results gathered. By the scheme of complementary nodes, even if a node has the shortest search path for the provider, it can still get information from its complementary node.

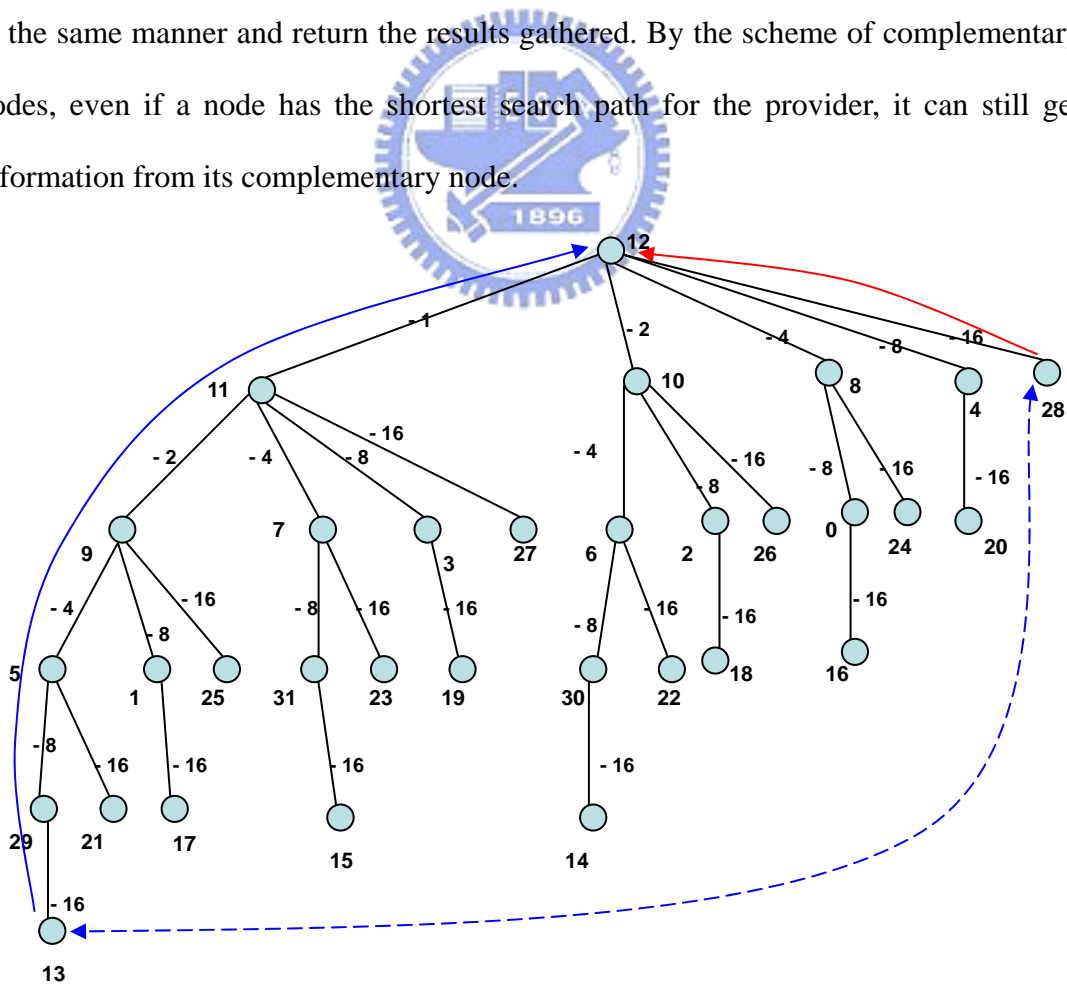


Fig.15: Requests to the complementary node for additional information

In Fig.15, *peer(28)* is the node with the shortest search path in the search tree for *peer(12)*. In the original scheme of reputation collection without complementary nodes, *peer(28)* can only get information from *peer(12)*. But if it sends requests to its complementary node, *peer(13)*, and ask for the scores stored along the search path of *peer(13)*, *peer(12)* will get the aggregated information from more node than the original condition.

3.3 Verification of Collected Information

Users search for the resources and contact resource managers to get the information of the resource providers. According to the aggregated scores listed in the data stored in the resource manager, users can directly choose one of the resource providers to get the wanted resources. But if the users do not trust the scores supplied by the resource manager or want to check for the truth of these supplied scores, we need a verification scheme for users to verify their got information.

3.3.1 Properties of Chord Search tree

As we describe in the previous sections, the structure of the search tree in Chord networks has the recursive property. Look at Fig.16 below. We can see that the 32-node search tree is divided into two 16-node trees by cutting the edge marked (-1). And the 16-node subtree can also be divided into two 8-node trees by the edge marked (-2). From these observations, we can conclude that: by cutting an edge marked (-2^k) , it will generate a subtree with $(N/2^{k+1})$ nodes, where N is the total number of nodes in the network.

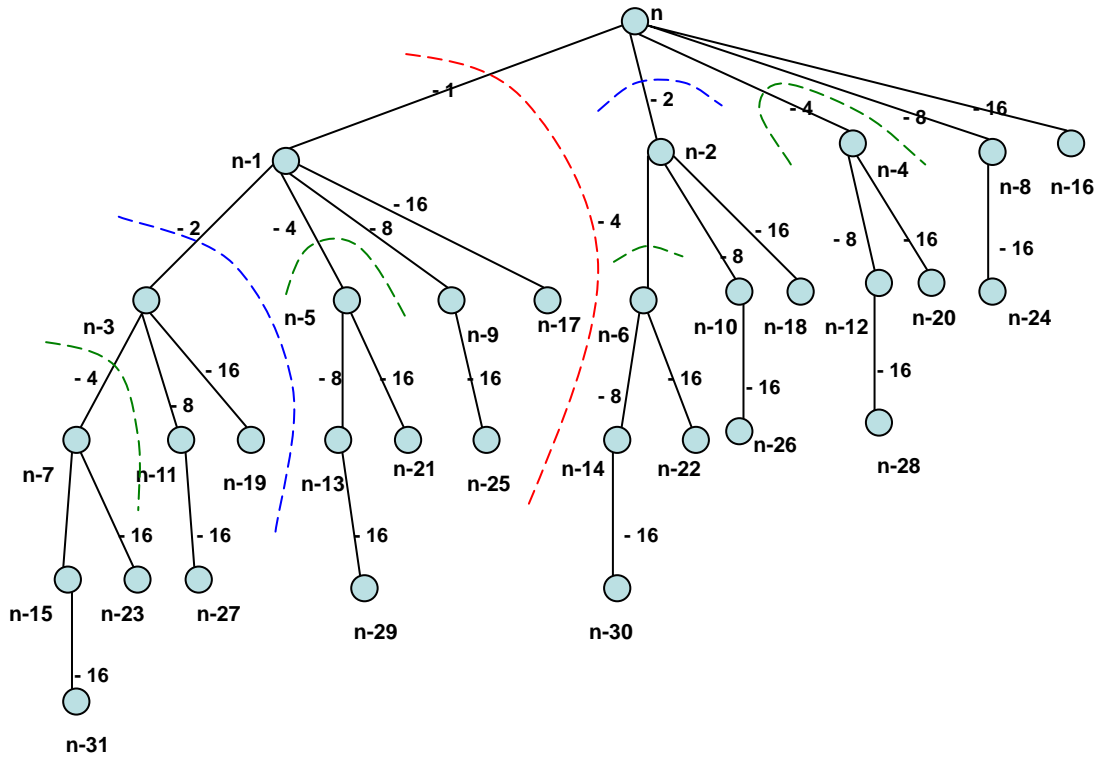
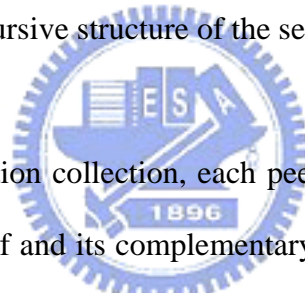


Fig.16: The recursive structure of the search tree in Chord



In our scheme of reputation collection, each peer can get the aggregated scores along the search paths of itself and its complementary node. The elements of the two search paths are two edges marked (-16) and edges marked (-8), (-4), (-2), (-1) respectively. According to the conclusion we make, the nodes below the edges marked (-16) must be the root of the subtree with $(N/32)$ nodes. In the case of Fig.16, the nodes are the roots of 1-node trees, which are leaf nodes. Similarly, the node below the edge marked (-8) is the root of the subtree with $(N/16)$ nodes, which is the root of 2-node tree.

3.3.2 Verification Scheme

In the scheme of recursive storage of reputations, the number of nodes which the aggregated value stored in a peer represents depends on the size of the subtree with

root is the peer. That is, the score stored in the root of 16-node tree represents twice as many nodes as the one stored in the root of 8-node tree.

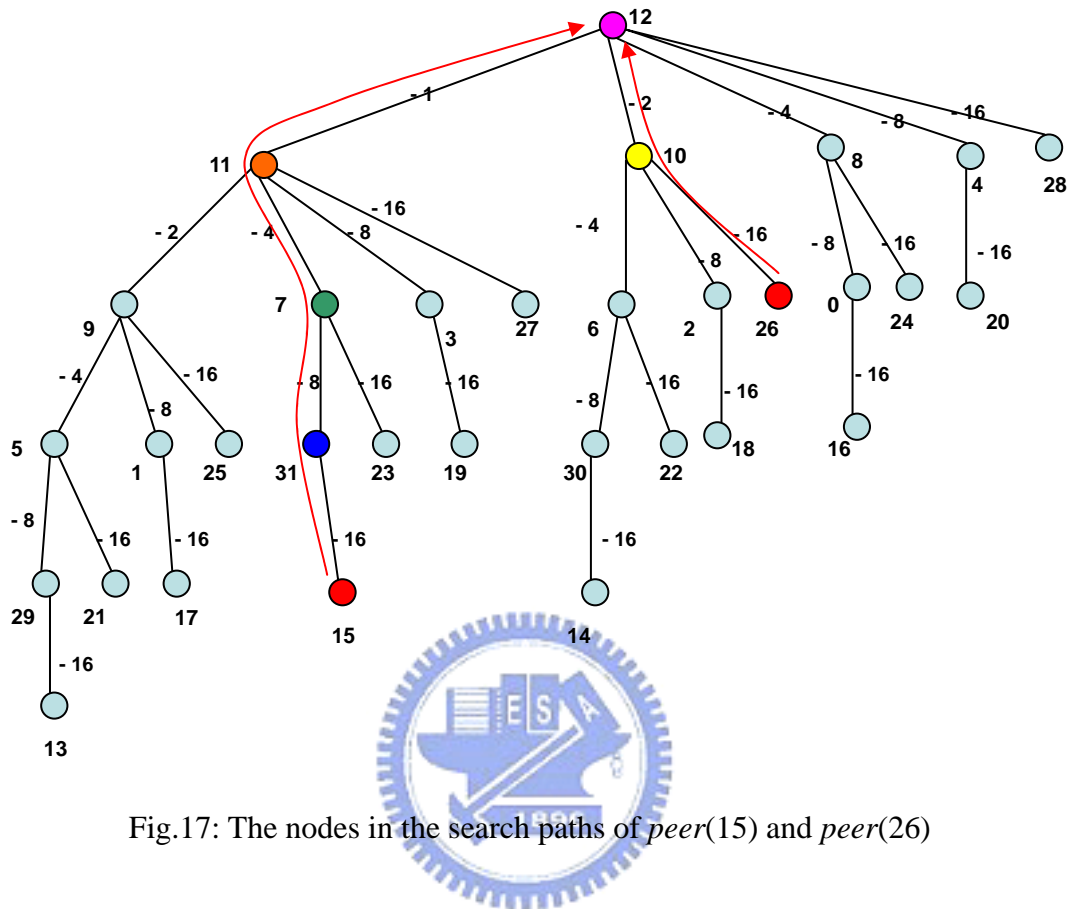


Fig.17: The nodes in the search paths of *peer*(15) and *peer*(26)

In Fig.17, we show the search paths of *peer*(15) and its complementary node, *peer*(26). And we mark the nodes along the two search paths by different colors. The two red nodes are leaf nodes, and the blue node is the root of 2-node tree. The green node is the root of 4-node tree while the yellow one is the root of 8-node tree.

After collecting the aggregated scores from these nodes, we can use the scores in the two red nodes to verify each other because the weight of the two scores should be the same. Then we use the two scores from red nodes to verify the score from blue node because the sum of the weight of the two red nodes should be equal to the one of the blue node. Because the sequence of the weight of these nodes can be treated as a geometric sequence, we can always merge the scores with smaller weights to verify

the score with heavier weight.

The scheme of verification can not only be used by general users, it can also be used by the search agents to verify the reputations they collect. Search agents collect all the aggregated scores along their search paths and the search paths of their complementary nodes. They can directly get information from the top s layers node in the two paths and then use other collected scores to verify the information.

3.4 Other Improvement of The Mechanism

3.4.1 Temporal Change

The environments of DHT-based peer-to-peer networks are dynamic, and the states of peers may change temporally. Suppose something happened to a peer so that it had poor performance before. Then the reputation scores assigned to it would be lower and the score about it listed in the resource manager may be lower than other providers. If the performance of this peer recovers suddenly, we hope that the scores in the resource manager can reflect the current condition as soon as possible. So we try to reduce the influence of the scores previously stored when computing the aggregating scores.

We set a time period (Ex: one day). And when we aggregate the scores getting from different time periods, we reduce the weight of the scores of the previous time period. For example, suppose there are time periods: t_{n-1} , t_n (as Fig.18). In the end of t_{n-1} , we aggregate the scores and get $R^{aggr}[t_{n-1}]$. We use $R^{aggr}[t_{n-1}]$ as the initial value of the time interval t_n to aggregate the scores received in t_n . When generating the aggregated scores of the time interval t_n , we reduce the weight of $R^{aggr}[t_{n-1}]$:

$$R^{aggr}[t_n] = \frac{R^{obv}[t_n] + f * R^{aggr}[t_{n-1}]}{1 + f}, \text{ where } 0 \leq f < 1$$

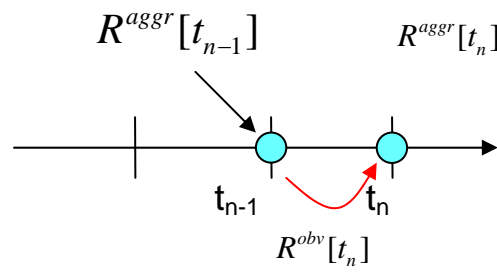


Fig.18: The aggregation of the scores in different time interval.



Chapter 4 Research Result

In this chapter, we will prove the scalability of our mechanism. We will also show some results of our research.

4.1 The Influences of Skip Distance

In our scheme of reputation storage, every peer uploads its reputation to the upper nodes along the search path. Peers determine which peers to upload according to the skip distance. After receiving the scores uploaded by other peers, each peer combines them and stores an aggregated score. The number of nodes which the aggregated score represents actually depends on the location of the peer in the search tree. In the search tree of a 2^m -node Chord network, every node is the root of a subtree with 2^k nodes, where k is between 0 and m . When $k=0$, this node must be a leaf node. And if $k=m$, the node must be the root node of the search tree. A peer receives the reputations uploaded by the nodes in the subtree whose root node is it. So the numbers of nodes which an aggregated score represents is directly proportioned to the size of the subtree whose root is the keeper of the score. Another factor which will influence the number of nodes which a aggregated score represents is the skip distance. In the trees with the same size, different skip distance will cause different storage conditions so that the number of node from which a peer will receive reputations will also be different. Next, we discuss the relation between the skip distance and the number of nodes from which a peer receives reputations.

4.2 The Relations between Search Trees and Pascal Triangle

By analyzing the searching trees of Chord, we can observe that each layer of the Pascal triangle corresponds to a searching tree with a specific number of layers. And the entries in one layer of the Pascal triangle give the numbers of nodes with different depths in the corresponding searching tree. The Fig.19 below is a part of the Pascal triangle where P_{ld} means the d -th entry in the l -th layer of the Pascal triangle:

P_{ld}	d						
	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4	6	4	1		
5	1	5	10	10	5	1	
6	1	6	15	20	15	6	1
⋮							

Fig.19: Pascal triangle.

As mentioned above, the l -th layer of the Pascal triangle corresponds to the searching tree with height = l (we define the height of a tree as the maximum distance from root to other nodes). And the d -th entry in this layer respectively gives the number of nodes with depth = $(l - d)$ in this searching tree. Now we take the searching tree with 32 nodes as example. The height of the searching tree of 32-node Chord network is 5. Fig.20 shows such tree structure. It also shows the number of nodes and the number of leaf nodes in each layer with different depths. We write the information in table format (Table.1). We can see that the numbers of nodes in layers with depths = $(5, 4, 3, 2, 1, 0)$ are $(1, 5, 10, 10, 5, 1)$ respectively, which is the same as the layer with $l = 5$ in the Pascal triangle.

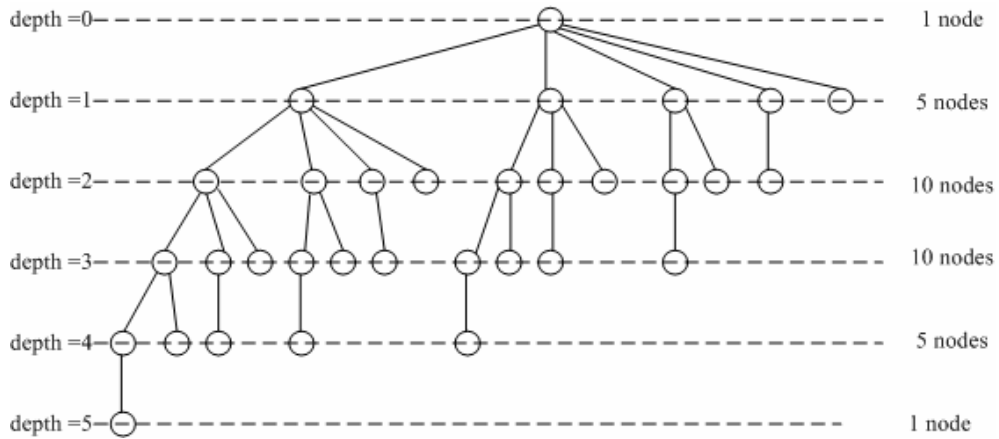


Fig.20: The number of nodes in each layer of the search tree

depth	Number of nodes
5	1
4	5
3	10
2	10
1	5
0	1

Table.1: The number of nodes in each layer of the search tree.

4.3 The Number of Nodes Which An Aggregated Score Represents

When skip distance = s , each peer will receive the reputations from the nodes which are $s, 2s, \dots$ layers lower than it in the search tree. In the previous sections, we mentioned that the number of nodes which are several layers lower than root is relational to the Pascal triangle. Therefore, when skip distance = s , the number of nodes from which a peer will receive reputations about a certain provider can be computed by summing some certain elements in the Pascal triangle. For example, if a node is the root of a 32-node Chord search tree, and suppose $s = 2$, we can get the value by computing the sum of the P_{50}, P_{52} , and P_{54} , which is $1 + 10 + 5 = 16$. From

this, we can know that when $s = 2$, the root of a 32-node search tree will receive the reputations from totally 16 nodes.

By continuously analysis, we compute the conditions under different values of network size and skip distance. When the size of identifiers is m , the size of the network is 2^m . Table.2 lists the numbers of nodes from which the roots of trees with different sizes m receive reputations under different value of skip distance s . And Fig.21 shows the ratio of the number of reputations a node receives to the number of the tree whose root is the node.

m	s=2	s=3	s=4	s=5	s=6	s=7	s=8
1	1	1	1	1	1	1	1
2	2	1	1	1	1	1	1
3	4	2	1	1	1	1	1
4	8	5	2	1	1	1	1
5	16	11	6	2	1	1	1
6	32	22	16	7	2	1	1
7	64	43	36	22	8	2	1
8	128	85	72	57	29	9	2
9	256	170	136	127	85	37	10
10	512	341	256	254	211	121	46
11	1024	683	496	474	463	331	166
12	2048	1366	992	859	926	793	496
13	4096	2731	2016	1574	1730	1717	1288
14	8192	5461	4096	3004	3095	3434	3004
15	16384	10922	8256	6008	5461	6451	6436
16	32768	21845	16512	12393	9829	11561	12872
17	65536	43691	32896	25773	18565	20129	24328
18	131072	87382	65536	53143	37130	34885	43912
19	262144	174763	130816	107883	77540	62017	76552
20	524288	349525	261632	215766	164921	116281	130816

Table.2: The result of computation under different values of m and s .

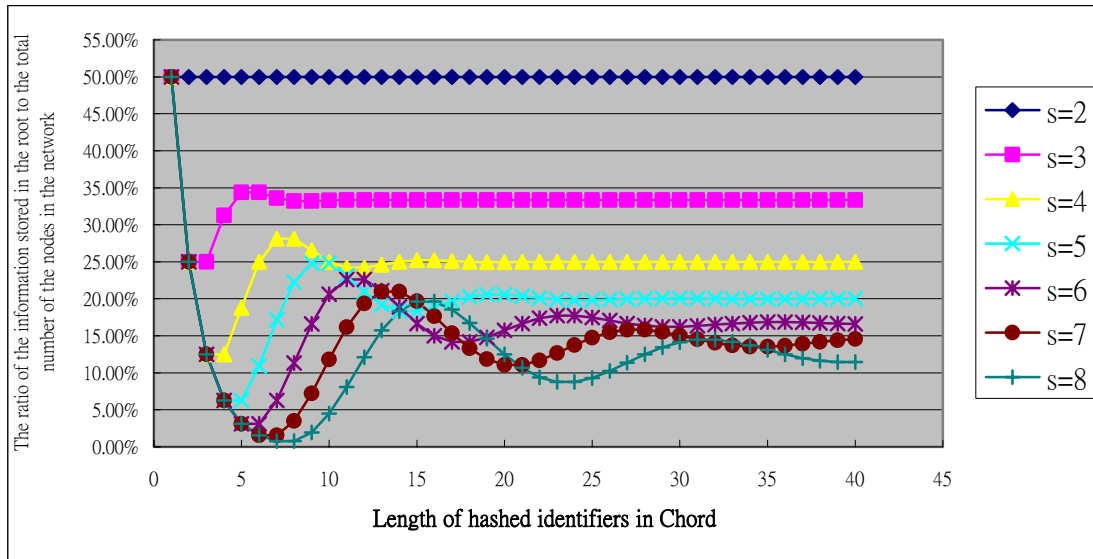


Fig.21: The ratio of the reputations stored in the root of a tree to the total number of nodes in the tree

With the increasing of the value of m , the ratio will converge to $(1/s)$. For example, when $s=8$, the ratio converges to $(1/8) = 12.5\%$. That is, the root of a tree will get reputation from $(1/8)$ of the nodes in the network. Therefore, the more the value of s is, the less each node has to maintain. But the number of nodes which the aggregated score can represent also becomes less than the conditions with smaller value of s .

4.4 The Information Can Be Collected in the Reputation Collection

As mentioned in section 3.2.3.2, the information contained in the whole search path can be got from the nodes in the top s layers of the search path. So the next to discuss is how much information which a user can get in the scheme of reputation collection.

In the section 4.3, we describe the information contained in the aggregated scores stored in a node. So we know that the root of the whole search tree will keep the score

containing the aggregated value of the opinions of (N/s) nodes. Taking the longest search path as example, the size of the subtree below a node is a half of the size of the subtree below its parent node. In Fig.32, $peer(n)$ is the root of the 32-node tree. while $peer(n-1)$, $peer(n-3)$, $peer(n-7)$, $peer(n-15)$, $peer(n-31)$ are the roots of the 16-node, 8-node, 4-node, 2-node, 1-node tree respectively.

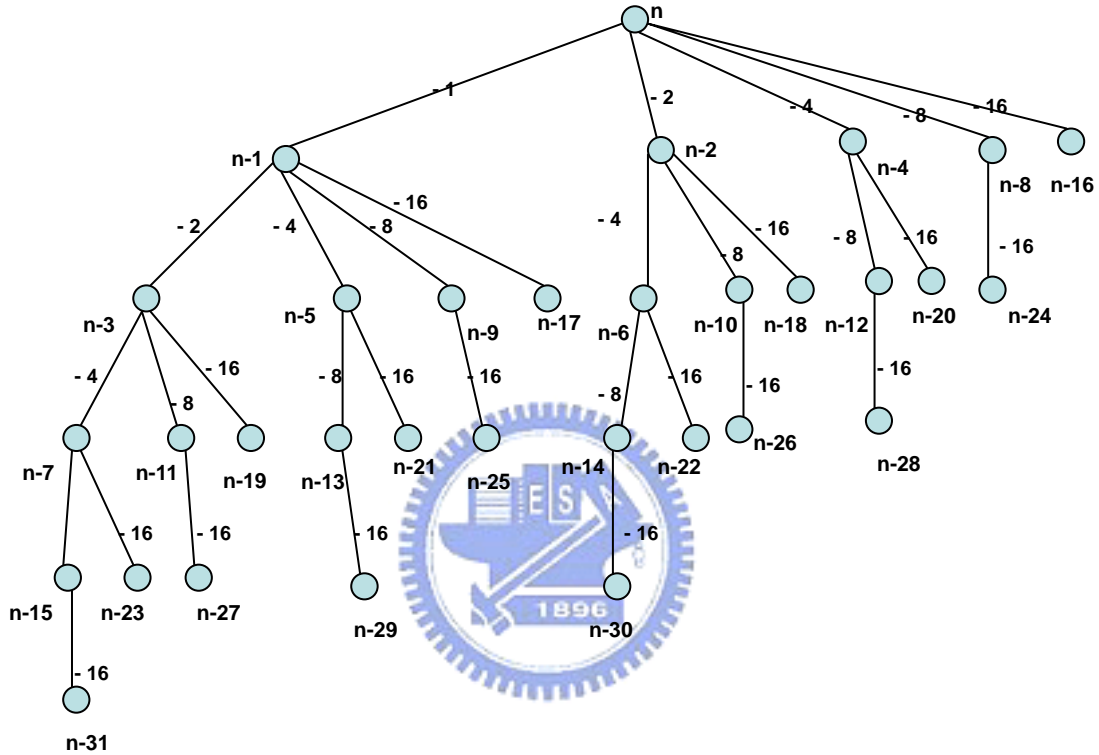


Fig.22: The search tree of a 32-node Chord network

So along the search path, the top node keeps the information from (N/s) nodes, and the second node keeps the information from $(N/2s)$. If we collect scores from the nodes in the top s layers of the search path, we can compute the amount of non-repeated information collected by:

$$\frac{N}{s} + \frac{N}{2s} + \frac{N}{4s} + \dots + \frac{N}{2^{s-1}s} = \left(\frac{2N}{s}\right)\left(1 - \frac{1}{2^s}\right)$$

When the size of the network is large enough, the information which we can get along the longest search path will converge to the aggregated opinions of $\left(\frac{2}{s}\right)\left(1 - \frac{1}{2^s}\right)$

of the nodes in the networks. Table.3 shows the result of the actual computation. We can see that the more the value of s is, the slower the ratio converges to $(\frac{2}{s})(1 - \frac{1}{2^s})$.

m	s=2	s=3	s=4	s=5	s=6	s=7	s=8
1	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
2	75.00%	75.00%	75.00%	75.00%	75.00%	75.00%	75.00%
3	75.00%	50.00%	50.00%	50.00%	50.00%	50.00%	50.00%
4	75.00%	50.00%	31.25%	31.25%	31.25%	31.25%	31.25%
5	75.00%	56.25%	31.25%	18.75%	18.75%	18.75%	18.75%
6	75.00%	59.38%	39.06%	18.75%	10.94%	10.94%	10.94%
7	75.00%	59.38%	46.88%	25.78%	10.94%	6.25%	6.25%
8	75.00%	58.59%	50.78%	34.77%	16.41%	6.25%	3.52%
9	75.00%	58.20%	50.78%	41.99%	24.61%	10.16%	3.52%
10	75.00%	58.20%	48.83%	45.61%	32.81%	16.80%	6.15%
11	75.00%	58.30%	46.88%	45.61%	38.96%	24.51%	11.13%
12	75.00%	58.35%	45.90%	43.24%	42.04%	31.59%	17.65%
13	75.00%	58.35%	45.90%	40.14%	42.04%	36.74%	24.54%
14	75.00%	58.34%	46.39%	37.63%	39.73%	39.32%	30.60%
15	75.00%	58.33%	46.88%	36.37%	36.27%	39.32%	34.94%
16	75.00%	58.33%	47.12%	36.37%	32.81%	37.24%	37.11%
17	75.00%	58.33%	47.12%	37.19%	30.22%	33.89%	37.11%
18	75.00%	58.33%	47.00%	38.27%	28.92%	30.12%	35.29%
19	75.00%	58.33%	46.88%	39.14%	28.92%	26.74%	32.21%
20	75.00%	58.33%	46.81%	39.57%	29.89%	24.30%	28.53%
21	75.00%	58.33%	46.81%	39.57%	31.35%	23.07%	24.88%
22	75.00%	58.33%	46.84%	39.29%	32.81%	23.07%	21.77%
23	75.00%	58.33%	46.88%	38.92%	33.91%	24.07%	19.58%
24	75.00%	58.33%	46.89%	38.62%	34.45%	25.68%	18.48%
25	75.00%	58.33%	46.89%	38.46%	34.45%	27.49%	18.48%
26	75.00%	58.33%	46.88%	38.46%	34.04%	29.12%	19.42%
27	75.00%	58.33%	46.88%	38.56%	33.43%	30.30%	21.02%
28	75.00%	58.33%	46.87%	38.69%	32.81%	30.89%	22.96%
29	75.00%	58.33%	46.87%	38.80%	32.35%	30.89%	24.90%
30	75.00%	58.33%	46.87%	38.85%	32.12%	30.41%	26.56%

Table.3: The ratio of the collected information along the longest path to the total number of nodes in the network

Fig.23 shows the graph of Table.3. We can see the convergence of the ratio under different values of s in the figure.

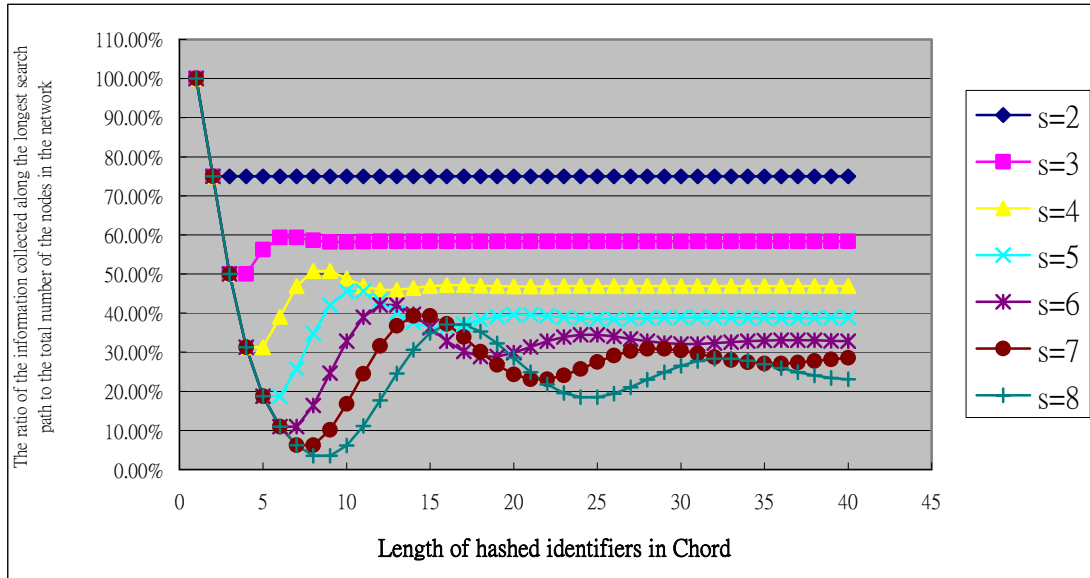


Fig.23: The ratio of the collected information along the longest path to the total number of nodes in the network

The complementary path of the longest search path is the shortest path in the search tree. So the information collected in the complementary path is only the aggregated score stored in the root node which has been gathered in the original path. That means that the complementary path does not aid reputation collection. But for other search paths, the nodes in the top s layers of their complementary paths will help to increase the information collected totally because the information additionally got is not repeated in the original path. So we can conclude that the information each peer can get in our approach represents the opinions of at least $(\frac{2}{s})(1 - \frac{1}{2^s})$ of all the nodes in the network when the network is large enough.

The amount of information each peer can collect in our mechanism only depends on the value of s . Therefore, we can conclude that: when the size of the network

doubles, the amount of information a peer can get will also double. That is, our algorithm is a scalable algorithm.



Chapter 5 Conclusion and Future Works

In this chapter, we will describe the conclusions of our research. Then we will also describe some future works of our works.

5.1 Conclusions

We present an algorithm for global trust and reputation management in DHT-based peer-to-peer networks. It includes the maintenance of the reputation scores and the aggregation of collected reputations. By the algorithm, each user can store the reputation scores assigned to a certain peer along the searching path for this peer. When users need some resources, they only need to search for the resource in the original manner. They can get the collected and aggregated scores supplied by the search agent and then make selection of resource providers directly according to the scores. If they do not trust the scores supplied by the search agents, they can also perform the verification scheme to verify the scores and make proper selection.

In our algorithm, peers in the DHT-based networks can get reputation scores in a fix ratio to the number of node in the network by contacting at most $\lceil \log_2 N \rceil$ nodes, where N is the total number of peers in the networks. In fact, when the network is large enough, peers can get at least the opinions of $(\frac{2N}{s})(1 - \frac{1}{2^s})$ nodes in the network, where s is the skip distance. The skip distance is a variable which can be determined in advance. So the mechanism in our approach is scalable.

5.2 Future Works

In our work, we focus on the management of reputation scores and trust relations in the DHT-based peer-to-peer networks. We don't limit the using of the reputation system. In fact, reputation systems can be multi-scoring. That is, users assign reputations in several manners. The method of assigning reputation scores influences the evaluation of reputation scores, and it also influences the precision of the decision making as a matter of course. To make the selection of resource provider more precisely, the assignments of reputations and trustworthiness are both worth researching. Additionally, we can also improve the verification scheme in our work. A more precise verification scheme can protect users from malicious behaviors more significantly.

The storage of reputations is another issue worth researching in this work. How to keep the convenience of reputation storage and collection and the distributed property of the DHT-based peer-to-peer networks is an important issue for us to continuously discuss.



Reference

- [1] E.K.Lua, J.Crowcroft, M. Pias, R.Sharma and S. Lim. “A Survey and Comparison of Peer-to-Peer Overlay Network Scheme”, In *IEEE Communications And Tutorial*, March, 2004.
- [2] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan. “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications” In *SIGCOMM’01, ACM*, August 27-31, 2001.
- [3] E. Damiani, S.D.C.d. Vimercati, S. Paraboschi, P.Samarati, F. Violante. “A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks” *CCS’02, ACM*, Washington, DC, USA. November 18-22, 2002.
- [4] R. Dingledine, N. Mathewson, P. Syverson. “Reputation in P2P Anonymity Systems”.
- [5] eBay website. www.ebay.com.
- [6] L. Xiong, L. Liu. “A Reputation-Based Trust Model for Peer-to-Peer eCommerce Communities”, In *Proceedings of the IEEE International Conference on E-Commerce (CEC’03)*, 2003.
- [7] B.K. Alunkal, I.Veljkovic, G.v. Laszewski, K. Amin. “Reputation-Based Grid Resource Selection”.
- [8] A. Josang, E. Gray, M. Kinateder. “Analysing Topologies of Transitive Trust”.
- [9] M. Richardson, R. Agrawal, P. Domingos. “Trust Management for the Semantic Web”.
- [10] S.D. Kamvar, M.T. Schlosser, H. Garcia-Molina. “The EigenTrust Algorithm for Reputation Management in P2P Networks”. *WWW2003, Budapest, Hungary*. May 20-24, 2003.