國立交通大學

網路工程研究所

碩 士 論 文

基於強韌搜尋之KAD同儕網路負載平衡方法

A Load Balancing Scheme for Resilient Search in KAD Peer-to-Peer Networks

研 究 生：吳岱庭

指導教授：王國禎　博士

中 華 民 國 九 十 八 年 六 月

基於強韌搜尋之KAD同儕網路負載平衡方法

A Load Balancing Scheme for Resilient Search in KAD Peer-to-Peer Networks

研　究　生：吳岱庭　　　Student：Tai-Ting Wu

指導教授：王國禎　　　Advisor：Kuochen Wang

國立交通大學
資訊學院
網路工程研究所
碩士論文

A Thesis
Submitted to Institute of Network Engineering
College of Computer Science
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science

June 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年六月

基於強韌搜尋之KAD同儕網路負載平衡方法

學生：吳岱庭　　指導教授：王國禎 博士

國立交通大學 資訊學院 網路工程研究所

摘 要

KAD 同儕網路已被廣泛地應用在檔案分享軟體中，但這些同儕網路仍就被不平衡的發佈負載所困擾，造成少量的節點處理大量的索引。這些高負載的節點會成為網路的瓶頸。因此，在本論文中，我們提出一個多次雜錯方法(KAD-N)以平衡網路中各節點的負載，其中 N 是一個由成本效益係數所決定的參數，它代表雜錯的最大次數。一個關鍵字經過 r 次的雜錯後，可以產生出一個發佈用的金鑰，而 r 是一個介於 1 與 N 之間的隨機數。模擬結果顯示，我們的方法的確可以將索引分佈的更平均。此外，針對我們的模擬環境，N = 7 (KAD-7)是最佳的設定。我們利用標準差來評估所提出的負載平衡方法。由模擬結果得知，KAD-7可以使搜尋命中率接近100%，其標準差也比KAD (即KAD-1)少了44%。這表示本方法的發佈負載比KAD更平衡，但它會增加 7% 的額外流量。當有節點失效時，本方法可藉由增加搜尋命中率加強搜尋的強韌性。此外，本方法可很容易地被延伸至其他以DHT為基礎的同儕網路。

關鍵詞：負載平衡，KAD，同儕網路，強韌搜尋。

A Load Balancing Scheme for Resilient Search in KAD Peer-to-Peer Networks

Student: Tai-Ting Wu **Advisor：Dr. Kuochen Wang**

Department of Computer Science

National Chiao Tung University

Abstract

Kademlia (KAD) peer-to-peer (P2P) networks have been widely used in file sharing applications. However, these P2P networks suffer from the unbalanced publishing load problem. It causes a few peers handling large numbers of indexes. Those peers with high loads may become the bottlenecks of the network. Therefore, we propose a *multiple hash* method (called KAD-*N*) to balance peer loads in the KAD network. Note that *N* is the maximum hash times, determining by a cost-effectiveness factor. This method hashes the keyword of an object *r* times to produce a key for publishing objects, where *r* is a random number and $1 \leq r \leq N$. Simulation results show that the distribution of indexes is more balanced using the proposed KAD-N method. We found out that $N = 7$ (KAD-7) is the optimal setting in our simulation environment. We used a standard deviation to evaluate the proposed load balancing method. Simulation results show that KAD-7 has the search hit rate close to 100% and the standard deviation is 44% less than that of the KAD (i.e., KAD-1), which means the proposed method is more load balanced than the KAD. However, KAD-7 has 7% extra traffic overhead. By increasing the search hit rate, KAD-N improves the search resilience of KAD networks with failed peers. Furthermore, the proposed KAD-N method can be easily extended to other DHT-based P2P networks.

Index Terms — Load balancing, KAD, peer to peer network, resilient search.

Acknowledgements

Many people have helped and encouraged me with this thesis. I appreciate my thesis advisor, Dr. Kuochen Wang, for his intensive advice and guidance. I would like to thank all the classmates in the *Mobile Computing and Broadband Networking Laboratory* (MBL) for their friendship and assistance. This work was supported by the National Science Council under Grants NSC96-2628-E-009-140-MY3 and NSC96-2628-E-002-138-MY3.

Finally, I thank my father and my mother for their endless love and support.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

The power to access useful information in a cost-effective manner is very important in today's Internet. Models for information sharing between users are currently being investigated for collaborative and cooperative media sharing applications. The most popular model to date is based on the P2P communication network model. A P2P network consists of peers that can play the roles of both clients and servers. Peers in a P2P network do not exploit an existing network infrastructure for operations such as routing and information retrieval. They use other peers' computing power and bandwidth to route and retrieve [1].

1.1 Types of P2P networks

Based on the structure of a network, P2P networks can be divided into three types: *structured*, *unstructured,* and *hybrid*. Most of the structured P2P networks are constructed by a distributed hash table (DHT). A peer in a DHT-based network represents a small database that is to store indexes of objects. When a peer intends to locate an object, it first needs to determine which peer storing indexes of the object. This is usually done by hashing an object's name to obtain a key that maps to a unique peer in the network. The remaining step then is simply routing the query message to the peer who storing indexes of the object [2]. Examples of DHT-based networks include KAD [3], Chord [4], Content Addressable Network (CAN) [5], Pastry [6] , and Tapstry [7]. They differ basically in how peers maintain their routing tables to guarantee an efficient route between peers. KAD has been widely used in file sharing applications. Famous applications, such as eMule [8] and BitTorrent [9], are both built based on KAD.

As to the unstructured type, peers are loosely coupled in an unstructured P2P network. The link between two peers is established in a more casual way. Peers only pay little cost to maintain the network. Unstructured P2P networks rely on flooding or random walk for searching, with query messages propagated to every peer. This results in increased network traffic [1]. Although unstructured P2P networks have some drawbacks, such as unguaranteed search and a large number of search messages; however, they are easy to implement and maintain. Several successful unstructured P2P systems have been deployed over the Internet, such as Gnutella [10], FastTrack [11], and iMesh [12].

JXTA-based P2P networks are a representative of a hybrid type. JXTA [13] is a set of P2P protocols which has been adopted by many applications, such as WiredReach [25] and Collanos [26]. It uses loosely-consistent DHT (LC-DHT) as the underlying query routing mechanism. The LC-DHT uses a hybrid approach that merges structured and unstructured P2P networks [14]. However, JXTA is lack of some functions, such as keyword search. Keyword search for JXTA has been developed in [15].

1.2 Comparison of search mechanisms

Structured P2P networks work more efficiently than unstructured P2P networks. Table I compares search mechanisms in these two kinds of P2P networks. The main difference between structured and unstructured P2P networks is where they store their indexes. Structured P2P networks store indexes in foreign peers. And they use a unique hash function to address the location of an index. Unlike structured P2P networks, unstructured P2P networks save indexes in peers who publish them. While querying a target, structured P2P networks use a hash function to find the unique location of the target. But unstructured P2P networks cannot know the target location. They send query messages to the network and hope peers who received query messages will propagate them to the target. And they use TTL (time to live) to restrict the maximum number of hops that a query message can reach. The default value of TTL is usually 7. This is commonly referred to as blind search [1].

Because of the search mechanism, unstructured P2P networks cannot guarantee that every object is searchable. They cannot find the object if it is in a peer who is more than 7 hops away. Structured P2P networks can search every object in the network because they use a hash function to locate. The overhead of searching an object in a structured P2P network is low because it does not flood query messages to the network. The search time complexity is *O(logN)* in structured P2P networks and *O(N)* in unstructured P2P networks (*N* is the number of peers in the network) [20].

TABLE I. Comparison of search mechanisms in P2P networks

Architecture	Structured	Unstructured
Indexes stored in	Foreign peer	Local peer
Query target	Unique	Blind
Guaranteed search	Yes	No
Search time complexity	*O(logN)*	*O(N)*

1.3 Motivation

In structured P2P networks, if we publish an object by keywords, popular keywords will produce a lot of identical keys. Each key has a specific target to publish. Target peers use these keys to construct indexes of objects. So the indexes of these popular keywords may aggregate in a few peers. Thus, it may cause unbalanced loads between peers. Peers who store the indexes of popular keywords consume more resources than others. In unstructured P2P networks, peers save their own indexes of published objects. They do not suffer from the above problem.

1.4 Proposed KAD-N

In this thesis, to resolve the unbalanced load problem in structured P2P networks, we proposed a KAD-N scheme that hashes the keyword of an object random times to produce a key for publishing the object. Other peers who want to publish the object with the same keyword will do the same. Different peers may hash different times to produce different keys, which all represent the same keyword. That is, our method will produce several different keys to represent a same keyword. These keys will be published to different peers, not only to one peer. And peers who received these publishing messages will use these keys to build indexes. Our method can spread indexes more even in the network. Simulation results show that we can balance the loads of peers and also increase the search hit rate in case that some peers failed or left. The overhead of our method is increasing the network traffic slightly, but not more than 10 percent.

The thesis is organized in the following manner. Chapter 2 is the preliminary knowledge of the KAD P2P network and related work. In Chapter 3, we describe our multiple hash method. Simulation setup and results are shown in Chapter 4. Chapter 5 discusses the implementation issues of the proposed KAD-N method. Chapter 6 gives some concluding remarks and future work.

Chapter 2
Preliminaries and Related Work

Since the KAD P2P network is our main target to enhance, in this chapter, we review this network and some existing load balancing methods. First, the distributed hash table (DHT), the main component of the KAD P2P network is overviewed. Then we describe the principles of how to lookup, publish, and search objects in the KAD P2P network. After that, we discuss the original KAD load balancing mechanisms. Next, other load balancing methods, such as Gossip Dissemination Strategy (GDS) [17], Rendezvous Directory Strategy (RDS) [18], and Independent Searching Strategy (ISS) [19], etc., are overviewed.

2.1 Distributed hash table

A main characteristic of DHT-based networks is that search is deterministic. It costs a bounded effort to route and retrieve. These DHT-based P2P networks basically support only the exact name match as each object is given a unique identifier obtained by hashing its name to determine its location in the network. Keyword search must be built on top of the overlay to enhance search functionality. Several mechanisms have been proposed for keyword search in DHT, and all of them use the inverted index as the primary data structure. An inverted index is a set of pairs (keyword, objects set). After an inverted index is built, we can use a keyword to find all objects that contain this keyword.

To implement keyword search in a structured P2P network, a distributed inverted index can be built. By using DHT-based networks, one can use a given keyword as a key to find out the peers who have objects that contain this keyword. Peers can retrieve objects with a given search query (keyword set) to perform a keyword search operation [2].

Object 1	Object 2	Object 3	Object 4
term 1	term 1	term 1	term 2
term 2	term 3	term 2	term 4
term 3	term 5	term 4	term 5

(a) Each object has three keywords.

Keyword	Objects set
term 1	{Object1, Object2, Object3}
term 2	{Object1, Object3, Object4}
term 3	{Object1, Object2}
term 4	{Object3, Object4}
term 5	{Object2, Object4}

(b) An inverted index.

Figure 1. An inverted index example of four objects [2].

In Figure 1(a), we show an inverted index example of four objects: objects 1, 2, 3, and 4. Each object contains three keywords. For example, object 1 has keywords, terms1, 2 and 3, and object 2 has term1, 3 and 5, etc. Figure 1(b) is the inverted index which is built by these keywords and objects sets. By this way, we can link keywords to different objects. For instance, if we use “term 1” as a keyword to search the network, objects 1, 2 and 3 will be found.

2.2 Background of KAD

KAD specifies the structure of a network and the exchange of information through peer lookups. Peers communicate among themselves in KAD using UDP [3]. A virtual overlay network is formed by the participating peers. Each peer is identified by a KAD ID. The KAD ID serves not only as an identification, but peers use the KAD ID to locate objects. In Figure 2, the KAD ID of the peer is “11111.” An object can produce two different keys, source keys and keyword keys. Source keys identify the content and location of an object. Keyword keys are computed by hashing a keyword from the name of the object [18]. In Figure 2, keywords of this object are “project” and “KAD.” Thus, the source key is “00110” and the keyword keys are “00010” and “11000.”

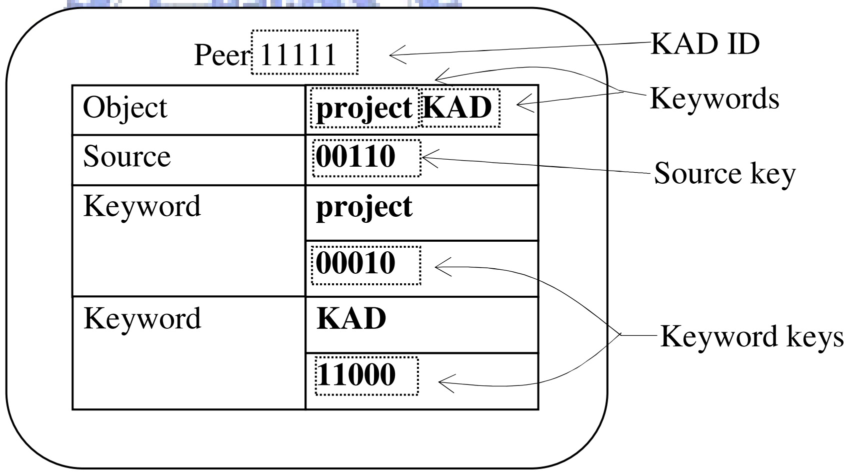


Figure 2. An example of a publishing peer.

Figure 3 andFigure 4 show an example of publishing an object. A peer wants to publish an object named “project KAD.” This object name will result in two keywords, “project” and “KAD.” All relevant references to the original object are generated, such as the source key and the keyword key. Next, keyword keys “project 00010” and “KAD 11000” are published to corresponding peers “00001” and “11001” to build indexes, which are all pointed to peer “00111.” Finally, the source key is published, with an index pointing to the publishing peer.

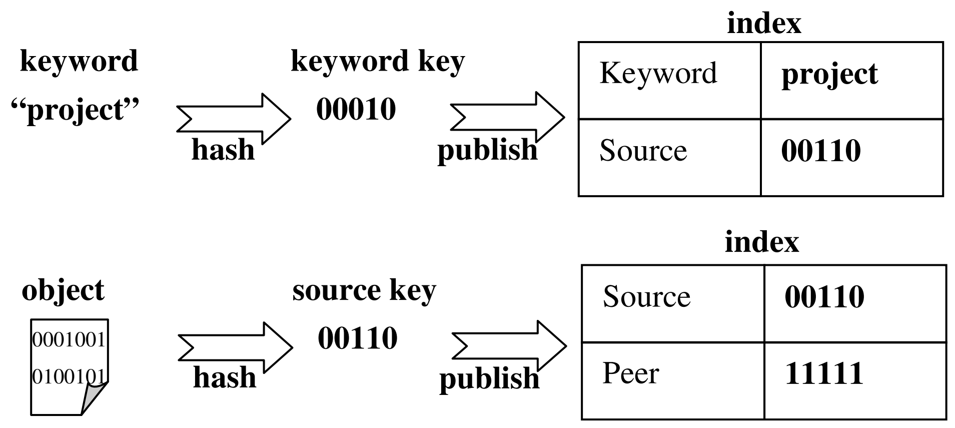


Figure 3. The index building procedure.

In KAD, each key is not published just on a single peer that is numerically closest to that key, but on 11 different peers whose KAD ID matches at least the first 8-bits of the key. This zone around a key is called the tolerance zone or the keyspace [16]. There are $2^8 = 256$ keyspaces in a KAD P2P network. In Figure 5, we can see 11 peers to receive publishing messages in the target keyspace.

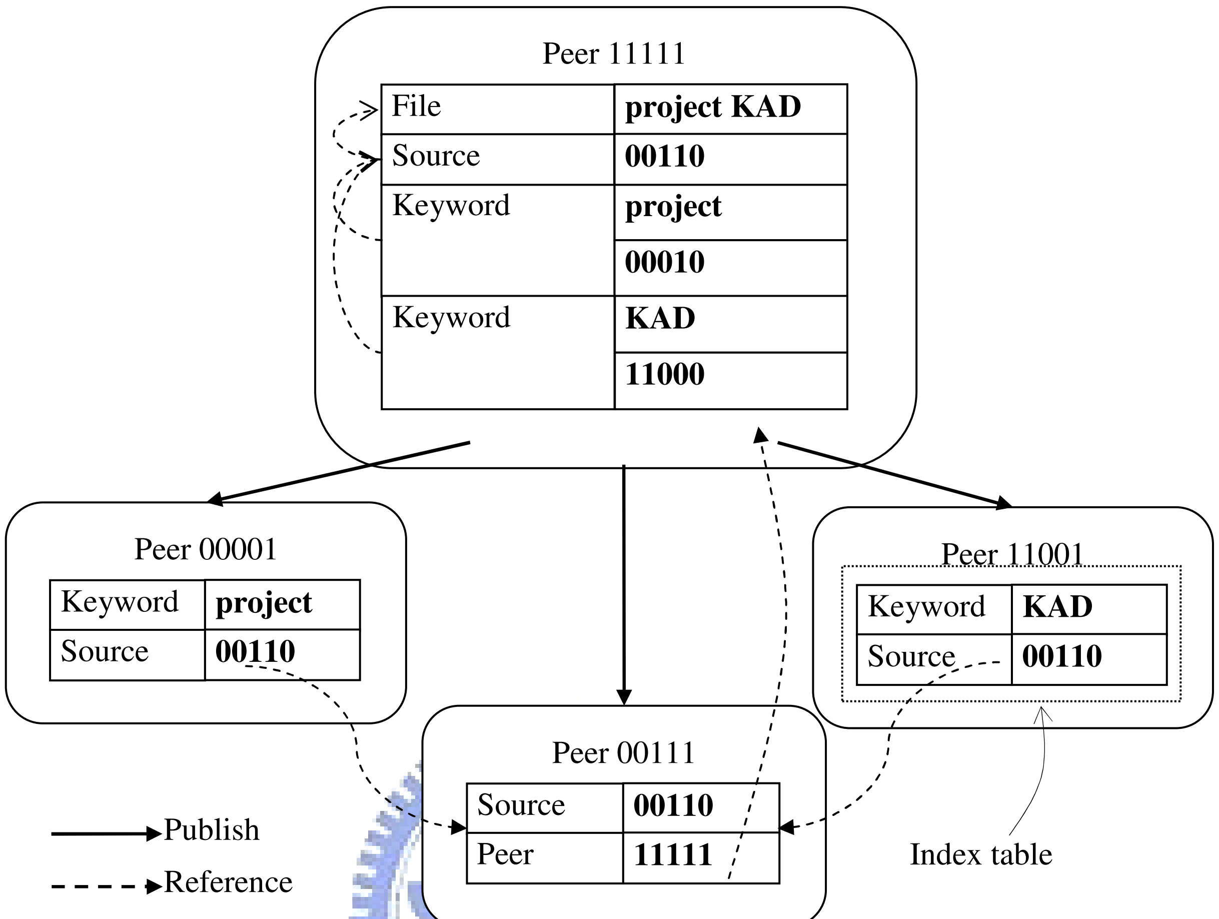


Figure 4. An example of publishing an object.

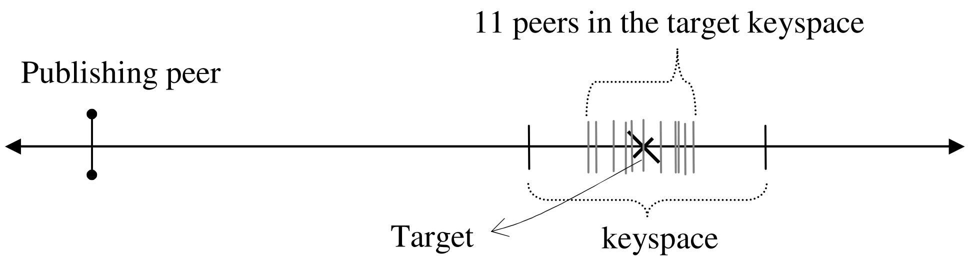


Figure 5. The diagram of where to publish [20].

When searching for some objects, the peer needs to know the target location and explores the network in several steps. Each step will find peers that are closer to the target. Figure 6 shows the steps of the lookup procedure. First, a searching peer sends messages to two closest possible peers. When the searching peer received responses, it obtains three more closer possible peers. Then these new possible peers that are in the target keyspace will be stored to a list called the candidate list. In this example, two peers are in the target keyspace. These two peers will be saved to the candidate list. In the last step, the searching peer sends a request for more closer peers to the three closest peers again, but only two peers are available and reply with closer peers. The lookup procedure terminates when the lookup responses contain only peers that are either already present in the candidate list or farther away from the target than the other top 3 candidate peers [16]. At this point, the candidate list is called stable. Like other DHT networks, KAD travels only *O(logN)* peers during the execution of the lookup procedure when there are *N* peers in the network. Therefore, the lookup procedure is very efficient.

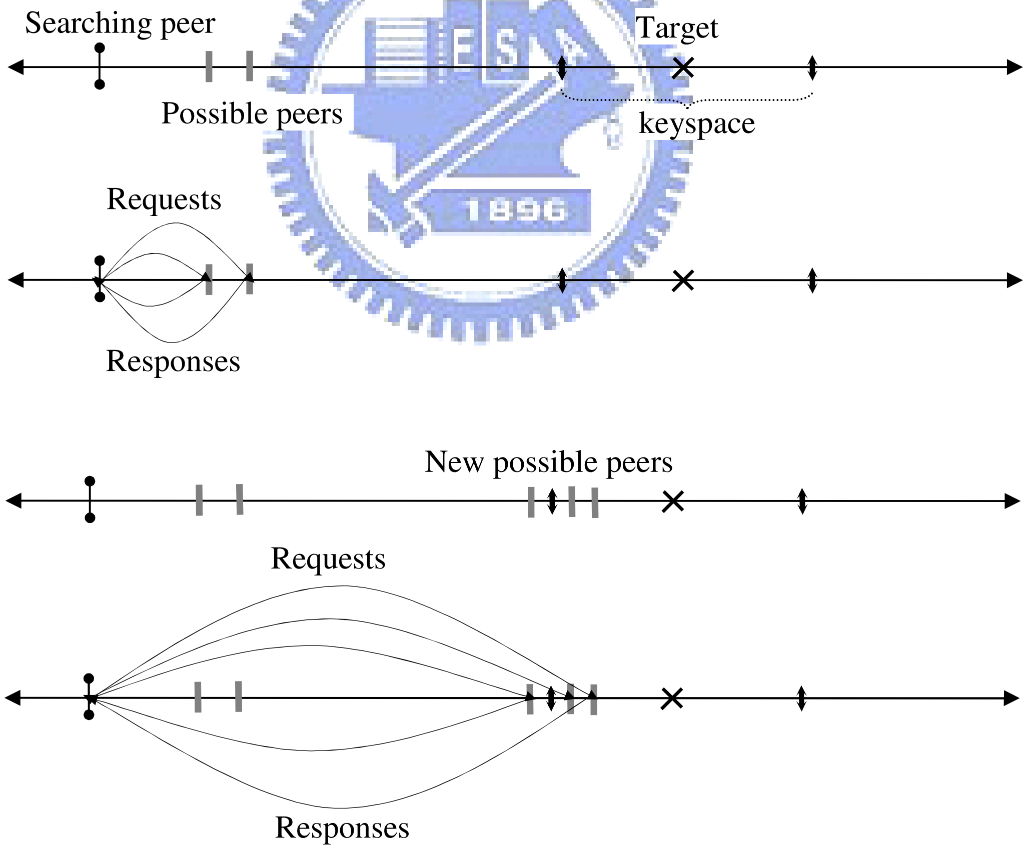


Figure 6. Steps of the lookup procedure [20].

2.3 Load balancing in KAD

KAD P2P networks do little to balance the load of each peer. They just limit the number of indexes handled in each peer to prevent them from overloading. A peer can only be responsible for maximum 60,000 indexes and can hold a maximum of 50,000 indexes of an individual keyword. Once reaching the limit, a peer would send an overload response to the publishing peer. After receiving an overload response, the publishing peer will publish this object to another peer.

2.4 Other load balancing methods

In RDS [18], the load information of each peer is periodically published to a few fixed rendezvous directories, which are responsible for scheduling the load reassignment to achieve load balance. Rendezvous directories are a fixed group of peers, which are known publicly to all peers. If a rendezvous directory is occupied by malicious peers or overwhelmed by DoS traffic, the service of load balancing failed.

In ISS [19], a peer doesn't publish its load information anywhere else. To achieve load balancing, peers should perform searching independently to find other peers with inverse load characteristics, and move load from the heavy nodes to the light nodes. ISS is very inefficient because searching peers with inverse load characteristics may incur huge traffic.

In [17], they proposed a GDS for load balancing in DHT based P2P networks, which combines RDS and ISS. The whole network is formed into groups, and the gossip protocol is used for load information dissemination. Each group member has the load information of its own group and the utilization information of the network after load information dissemination. They use these load information to achieve load balancing. GDS can only be built on top of ring-like DHT based P2P networks, such as Pastry, Chord, etc.

In [27], they presented a dynamic feedback adaptive scheduling algorithm to adjust the ratio peers distribution constantly according to the super peers' cyclical feedback information, which fully taking account the load capacity of each super peers, and solving the problem of super peers load balance in hybrid P2P networks effectively [27]. The super peers in a system are divided into several cluster networks according to their different locations in the network, and each cluster network communicates with load balancing control peer through a cluster agent peer. All the requests from peers should be first sent to the load balancing control peer, which designates logging super peers for peers according to the immediate updated super peer scheduling sequence. There may be a number of this kind of load balancing control peers in the system [27].

Chapter 3
Design Approach

In general, each keyword of an object will be hashed one time to get a key to publish. To enhance the search hit rate, we propose a KAD-N method to publish keywords by multiple hash in the KAD P2P network, where *N* is the maximum hash times. By hashing the key of a keyword, we can get the second key that can represent the same keyword. We can still hash the second key to get the third key. By hashing it random times, we get a final key to publish. Peers may produce different keys to represent the same keyword. And different keys are published to different peers. This multiple hash method will publish a keyword from peers to different peers to balance their loads.

3.1 Concept of KAD-N

Figure 7 shows the procedure of how to generate a key to publish. If we want to publish object 1, we will use terms 1, 2, and 3 as keywords. We use term 1 as an example to describe our multiple hash operation. First, we generate a random number *r* and $1 \leqq r \leqq N$. Next, we hash term 1 to get key 1 and hash key 1 to get key 2, etc. After *r* times of hashing, we get key *r*, which is the final key to publish. Then we publish key *r* to the network.

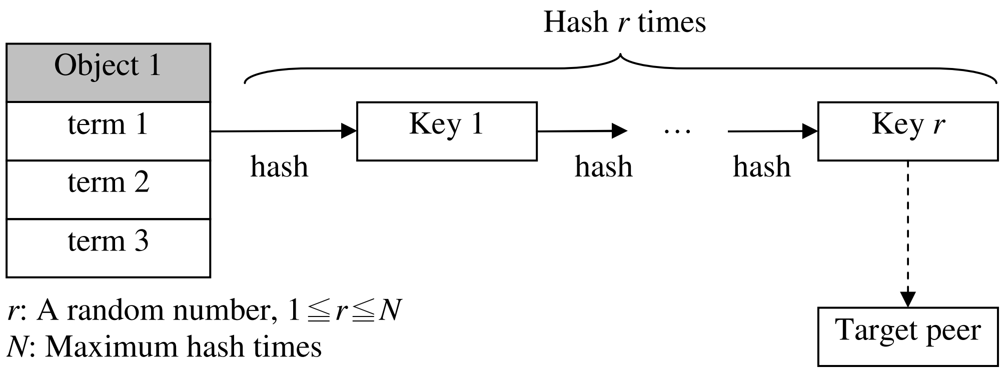
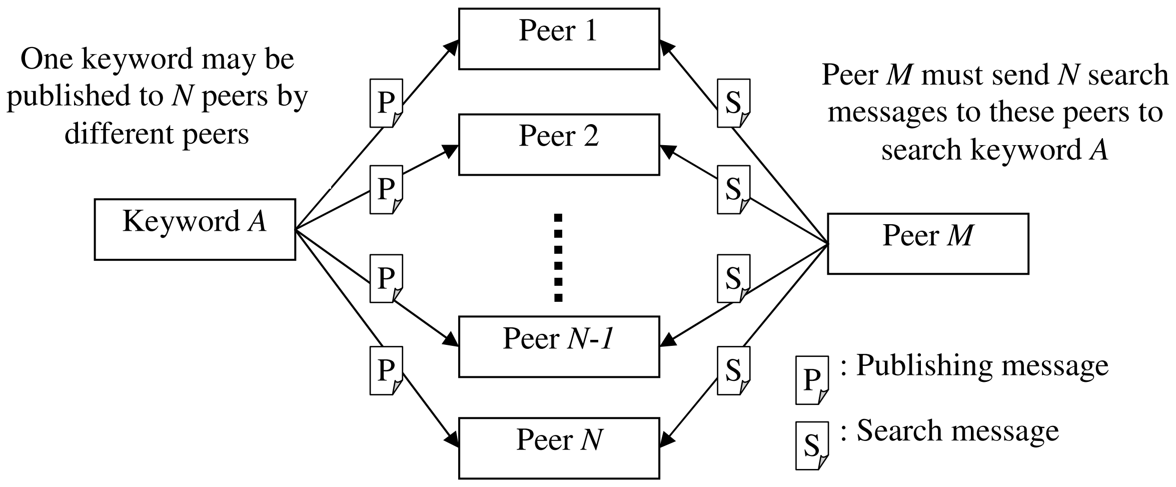


Figure 7. The procedure of how to generate a key to publish.

However, the multiple hash will increase the number of search messages. If we publish a keyword by hashing the key twice then we must query two keys in average to get the whole indexes of a keyword. Figure 8 shows the concept of multiple hash for publishing and search. In this figure, keyword *A* may be published to peers 1 to *N*. If peer *M* want to search keyword *A,* it must send *N* search messages (queries) to these peers who may store indexes of keyword *A*. Then peer *M* will receive *N* search responses. It can then build the indexes of keyword *A* from these responses. Therefore, the proposed KAD-N method will result in more traffic in searching keywords. However, the number of search messages is usually ten times less than publishing messages [16]. Hence, KAD-N requires just a small overhead to balance the publishing load.



One keyword may be published to *N* peers by different peers
Keyword *A*
P
Peer 1
Peer 2
Peer *N-1*
Peer *N*
S
Peer *M* must send *N* search messages to these peers to search keyword *A*
Peer *M*
P : Publishing message
S : Search message

Figure 8. The concept of multiple hash for publishing and search.

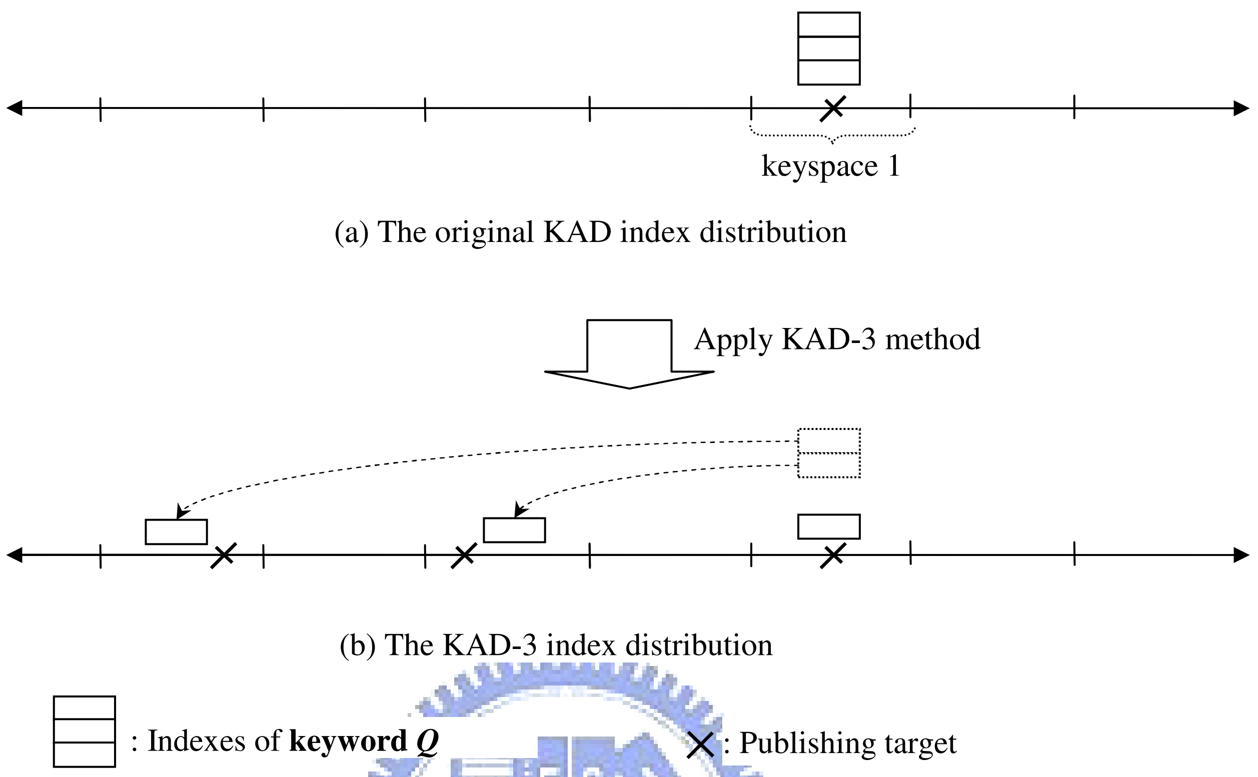

keyspace 1
(a) The original KAD index distribution
Apply KAD-3 method
(b) The KAD-3 index distribution
: Indexes of **keyword *Q***
: Publishing target

Figure 9. The concept of multiple hash for balancing the index distribution.

Figure 9 is an example of multiple hash for balancing the index distribution. Figure 9(a) is the index distribution of the original KAD. All indexes of an individual keyword will be handled by a keyspace, e.g. keyspace 1 handleing all indexes of keyword *Q*. After applying the proposed KAD-N method to the network, the index distribution will spread more even. In this example, we apply KAD-3 to the network. KAD-3 will distribute indexes of keyword *Q* into 3 keyspaces as shown in Figure 9(b). One is handled by the original keyspace 1 and the other two will be spread to other two keyspaces.

3.2 Publishing procedure

The detailed publishing procedure of KAD-N is shown in Figure 11. Remind that *N* is the maximum hash times. A KAD-N network will work as the original KAD P2P network if *N* is set to 1. In Chapter 4, the optimal hash times will be derived. Note that, we will discuss block A1 of Figure 11 in Chapter 5.

To publish a file, first, we get publishing keywords from the object name. Then a random number *r* will be generated for each keyword (for example, keyword *A*) and $1 \le r \le N$. After that, we hash keyword *A* *r* times to produce the final target key: key *a*. Figure 10 shows the proposed multiple hash algorithm. Key *a* will be the target while running the lookup procedure. The details of the lookup procedure were mentioned in Chapter 2. After starting the lookup procedure, we will receive several responses which contain some possible peers who are closer to the target. We use these peers to update the candidate list. If the candidate list becomes stable, then advance to the next step. The candidate list will be sorted by the distance to the target in the ascending order. The closest node is the first one in the list. In Chapter 2, we mentioned that KAD will publish a key to 11 different peers in the target keyspace. That is, top 11 nodes will be selected from the list for publishing. After sending publishing messages to these nodes, we successfully publish a keyword to the KAD P2P network.

```
Multiple Hash Algorithm
IUPUT
  keyword A:          /* keyword to be hashed;
   N:                 /* maximum hash times;

OUTPUT
  key a:              /* key to be published;

ALGORITHM
  r = random(1, N)
  i = 1
  tmp = keyword A
  While i <= r Do
    tmp = hash(tmp)
    i = i+1
  end While
  key a = tmp
  Return key a
```

Figure 10. Multiple hash algorithm.

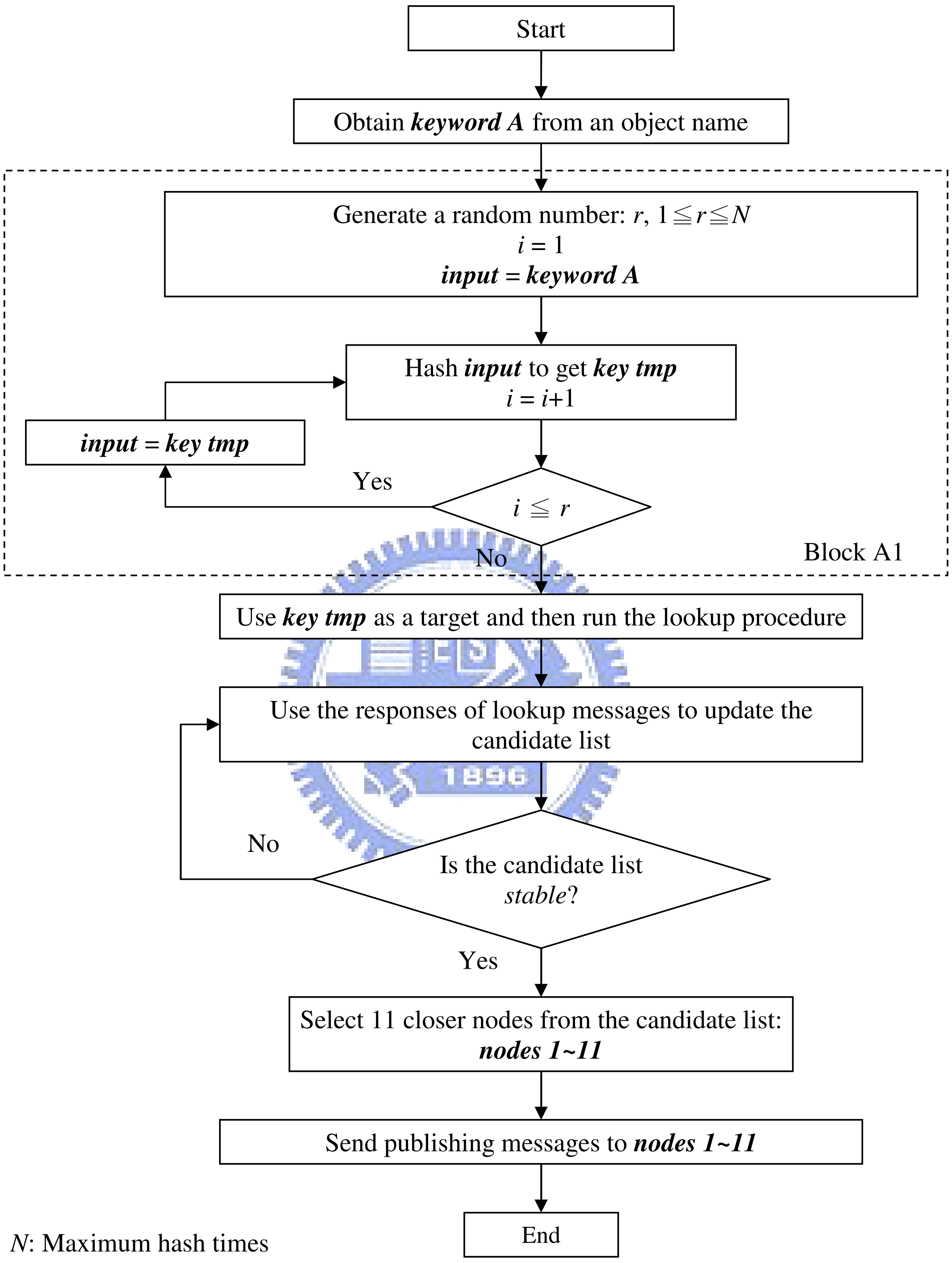
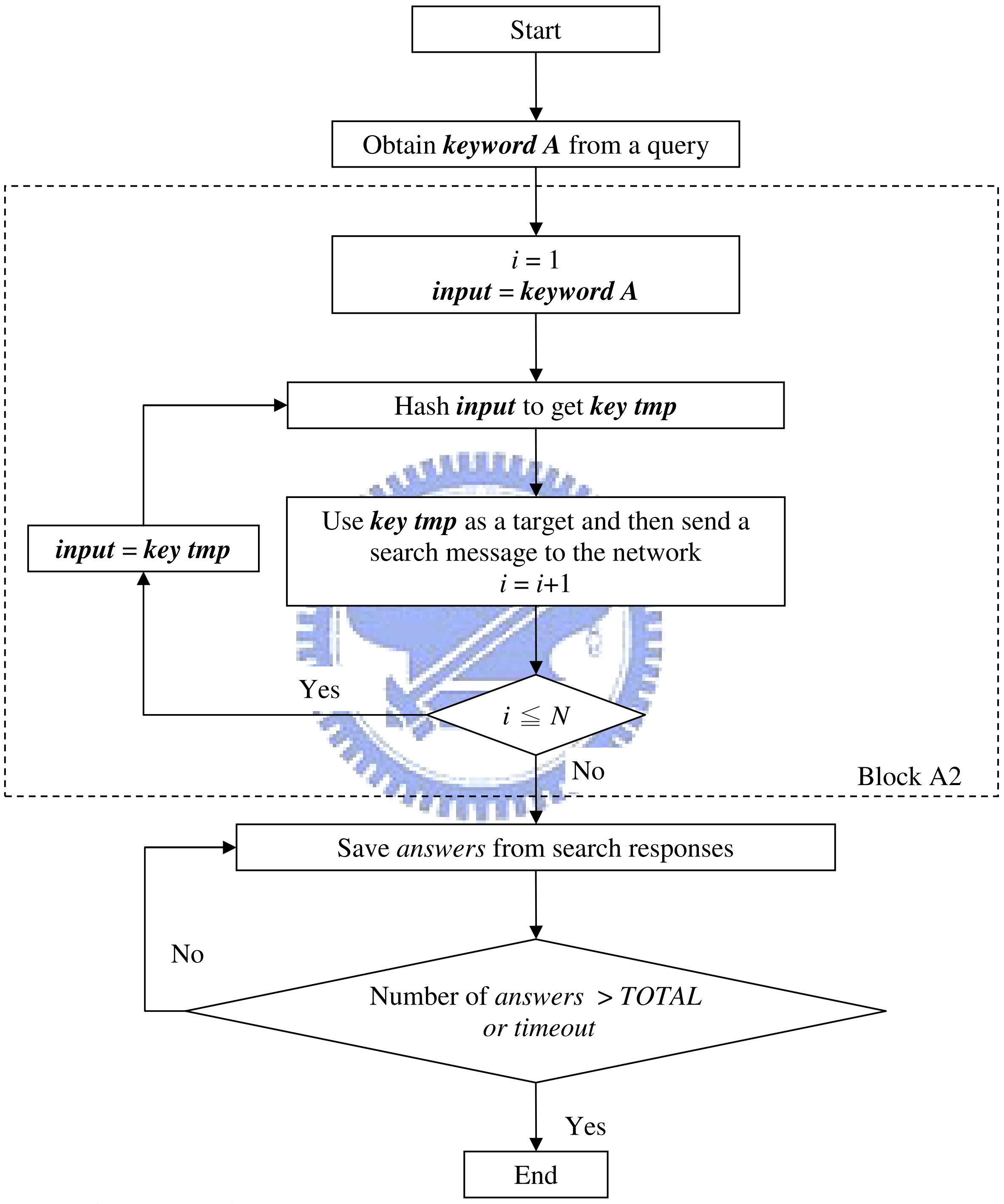

Start
Obtain keyword A from an object name
Generate a random number: r, 1≦r≦N
i = 1
input = keyword A
Hash input to get key tmp
i = i+1
input = key tmp
Yes
i ≦ r
No
Block A1
Use key tmp as a target and then run the lookup procedure
Use the responses of lookup messages to update the candidate list
No
Is the candidate list stable?
Yes
Select 11 closer nodes from the candidate list: nodes 1~11
Send publishing messages to nodes 1~11
End
N: Maximum hash times

Figure 11. The publishing procedure of KAD-N.

3.3 Search procedure



Start
Obtain ***keyword A*** from a query
i = 1
input* = *keyword A
Hash ***input*** to get ***key tmp***
Use ***key tmp*** as a target and then send a search message to the network
i = *i*+1
input* = *key tmp
Yes
i ≦ *N*
No
Block A2
Save *answers* from search responses
No
Number of *answers* > *TOTAL* *or timeout*
Yes
End

N: Maximum hash times
TOTAL: Maximum number of answers

Figure 12. The search procedure of KAD-N.

Figure 12 describes the search procedure of KAD-N. At the beginning, we obtain a keyword from a query (for example, keyword *A*). After that, this keyword will be hashed to produce a temp key. We use this temp key as a target to generate a search message and send it to the network. The above procedure will be repeated *N* times. That is, it will send *N* different search messages to the network. Then we will receive several search responses which may contain search answers. The search will stop when we receive enough answers or timeout is triggered. The default *TOTAL* value is 300 and the default timeout is set to 20 seconds. In other words, we will stop the search process after 20 seconds or if we receive more than 300 answers [18]. We will discuss block A2 of Figure 12 in Chapter 5.

3.4 Qualitative comparison of KAD-N and KAD

Table II shows the comparison between the original KAD and our KAD-N. If we hash a keyword at most *N* times, the publishing load will be more balanced and the search hit rate will also increase. Indexes of a keyword are published to at most *N* targets. Furthermore, KAD-N does not increase the total number of indexes. It just distributes indexes more even, as shown in Figure 9. In other words, in KAD-N the total publishing load of the network is same as that of KAD but the number of search messages will increase *N* times compared to KAD. Because KAD-N will spread the indexes, there will be more peers who have the same indexes. KAD-N will improve the search hit rate in case that some peers failed. However, the network traffic will increase slightly because of the increased number of search messages. The keyword of an object may be hashed at most *N* times and the computation overhead is thus *O(N)*. In the original KAD, since each keyword is hashed only once, the computation overhead is *O(1)*. KAD-N would cause extra computation overhead. We will discuss an optimal value of *N* in the following chapter.

TABLE II. Qualitative comparison of KAD and KAD-N

Approach	KAD	KAD-N (proposed)
Publishing load	Imbalance	Balance
Search hit rate	Normal	Better
Computation overhead	*O(1)*	*O(N)*
Query messages per search	1	*N*
Network traffic	Normal	More

Chapter 4
Simulation Results

4.1 Simulation setup

First, we analyze the overhead of publishing messages and search messages in KAD. In [21], they spied on 20 different keyspaces of the KAD network for 24 hours. During this time, on average, 4.3 million publishing messages and 350,000 search messages were recorded. Based on the measurements of [21], it showed that there are ten times more publishing messages than search messages. Moreover, a publishing message is ten times bigger than a search message since it contains not only a keyword but also metadata describing a published object. In [23], they also spied on a keyspace in the KAD P2P network for 12 hours. They got 561,542 search messages and 5,549,183 publishing messages. Search messages produced 10.8 MB traffic and publishing messages produced 966 MB traffic. Based on these data, traffic produced by a search message is 0.019 KB and 0.18 KB for a publishing message on average. We used these data to calculate total network traffic in our simulation environment. Total network traffic contains search traffic and publishing traffic.

We rank keywords according to their appearance times. Rank 1 is the most popular keyword. The publishing messages, which were collected by [21], contain 26,500 different keywords per keyspace and 315,000 distinct files. The appearances of each keyword were also counted. Based on these data, [21] used Matlab to estimates the number of indexes for the i'th popular keyword which is proportional to $1/i^{1.63}$ and the number of indexes for the most popular keyword is about 10^7. For example, the number of indexes of the most popular keyword is ten times more than the tenth popular keyword in the KAD P2P network. That is to say, the peer who handles indexes of the most popular keyword will get ten times network load than the peer who handles indexes of the tenth popular keyword. Based on the above analysis, we evaluate the performance of our approach.

We used JAVA to construct our simulation environment. Based on [20] and the above analysis, we simulated the behaviors of how KAD P2P networks publish objects and distribute indexes. The indexes handled by each peer were also recorded. Then we applied our method to this simulation environment. We gathered the indexes handled by each peer and used them to show the effectiveness of the proposed KAD-N method.

4.2 Simulation results

Figure 13 shows the index distributions of each keyspace under different hash times. We rank keyspaces according to the number of indexes handled, i.e. keyspace popularity. Rank 1 keyspace handles the most indexes. We found that the index distribution of the original KAD (KAD-1) is very uneven. A large number of indexes were handled by a few keyspaces. If we hash more times, some indexes would be moved from front rank keyspaces to others. From this figure, the index distribution curve will be smoother if we hash the key more times. In other words, if we hash the key more times, publishing load will be more balanced.

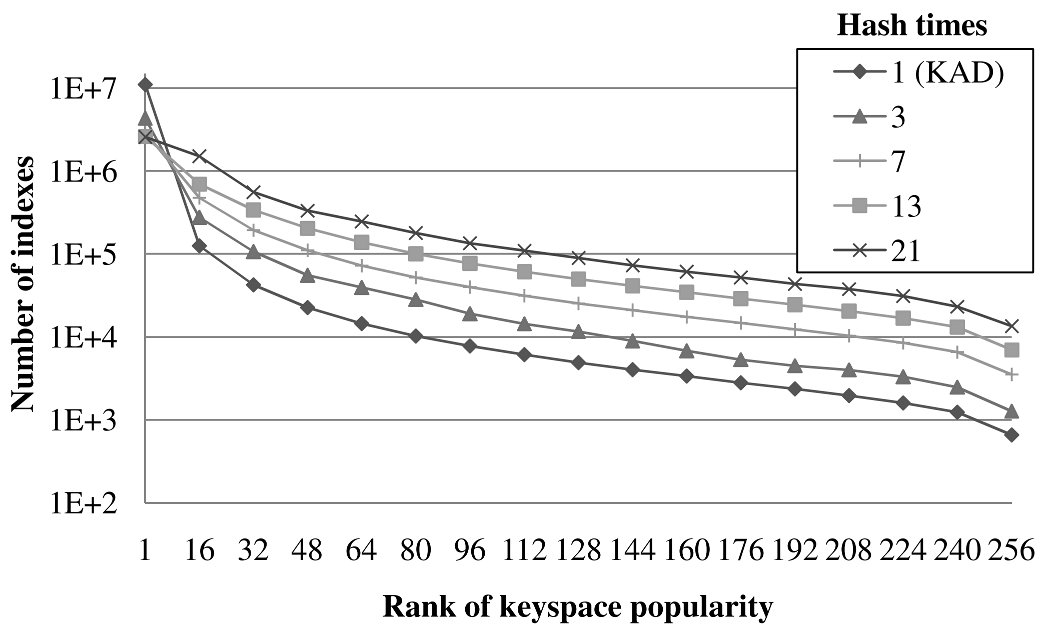


Figure 13. The index distribution of each keyspace under different hash times.

However, the number of search messages will increase after applying the proposed KAD-N method. We found that the total network messages will increase linearly with more hash times in Figure 14. Total network messages were calculated based on [23], which was introduced in the simulation setup. They include search messages and publishing messages.

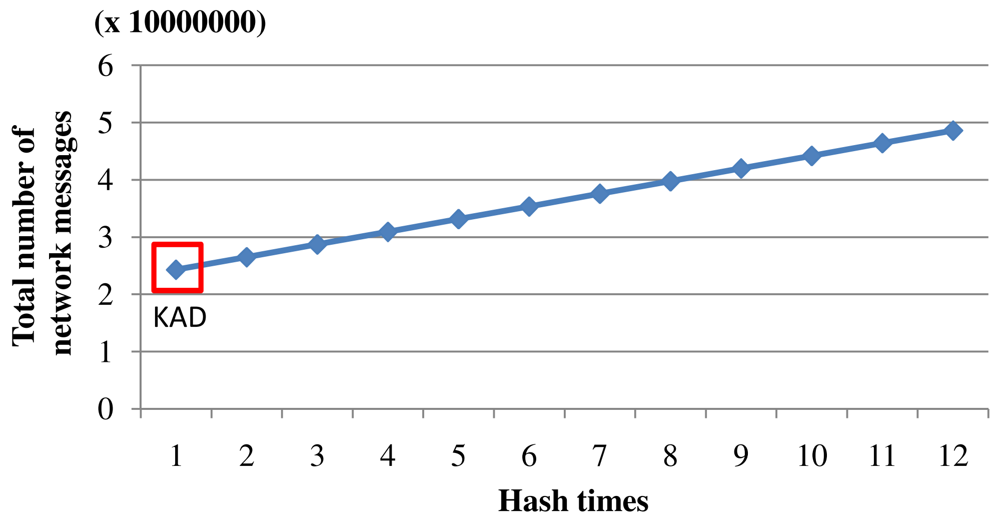


Figure 14. The total number of network messages under different hash times.

Figure 15 plots the percentage of extra traffic under different hash times. The growth of the curve is linear just like Figure 14. As we mentioned in the simulation setup, number of search messages multiplied 0.019 KB is search traffic and number of publishing messages multiplied 0.18 KB is publishing traffic. We add search traffic to publishing traffic to get total network traffic. We calculate extra traffic percentage p according to the following equation:

$$p = \frac{increased\ search\ traffic}{total\ network\ traffic\ of\ original\ KAD}$$

The extra traffic is very small because the number of search messages is much fewer than the number of publishing messages, and traffic produced by a search message is much smaller than a publishing message.

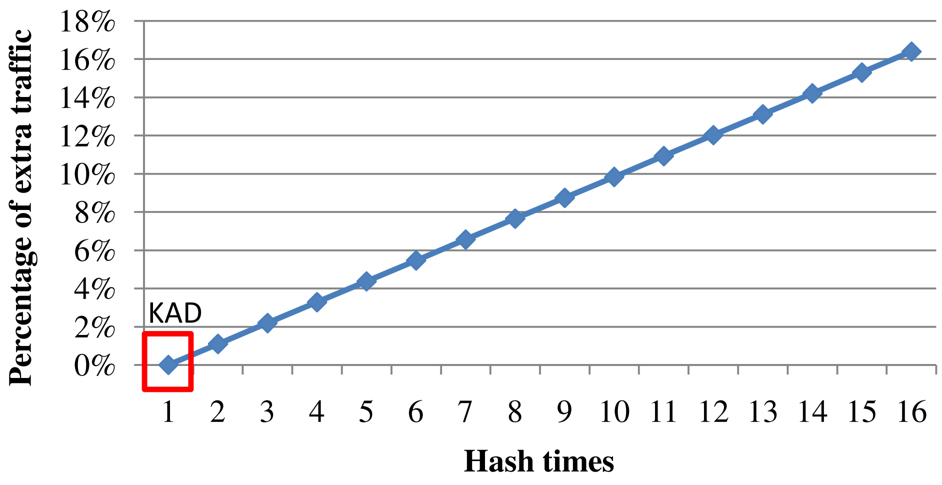


Figure 15. The percentage of extra traffic under different hash times.

We use a standard deviation σ to show the divergence under different hash times. A standard deviation is a measure of the dispersion of a data set. A low standard deviation indicates that the data points tend to be very close to the mean, while a high standard deviation indicates that the data are “spread out” over a large range of values. We calculate standard deviation using the number of indexes handled by each keyspace. In other words, the higher the value, the more unbalanced publishing load of each keyspace. σ is computed as follows:

$$\boldsymbol{\sigma} = \sqrt{\frac{\sum_{i=1}^{n}(\boldsymbol{X_i} - \boldsymbol{\mu})^2}{\boldsymbol{n}}}$$

where n is the number of keyspaces (n = 256 in KAD); X_i is the number of indexes handled in the ith keyspace and μ is the average number of indexes handled in each keyspace.

From Figure 16, we observed that when hash times ≥ 7, σ will not decrease too much. That is, if we hash more than 7 times, the standard deviations are almost the same. In other words, when hash times ≥ 7, it doesn’t help much on load balancing.

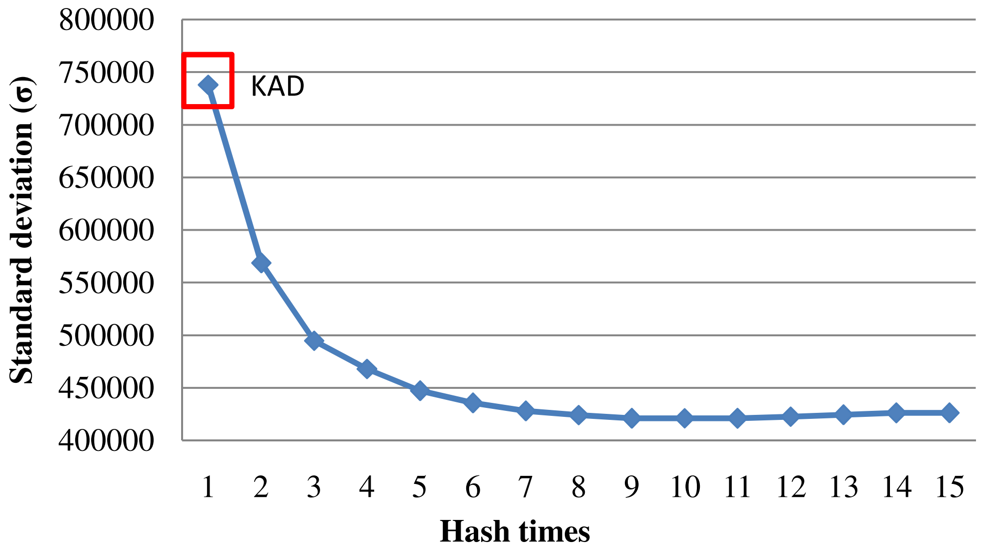


Figure 16. The standard deviation of each keyspace under different hash times.

We also simulated the hit rate variation under different hash times in case that some peers failed. We cannot retrieval indexes from failed peers. We call these indexes as missing indexes. Objects referenced by missing indexes would be unsearchable. Note that the hit rate is calculated by the number of missing indexes dividing number of total indexes. In Figure 17, by hashing more times it increases the hit rates while peers failed. From [22], we know the number of peers vary according to a diurnal. And the minimal number of peers is about 78% of the maximum. So the percentage of failed peers in a day is about 27%.

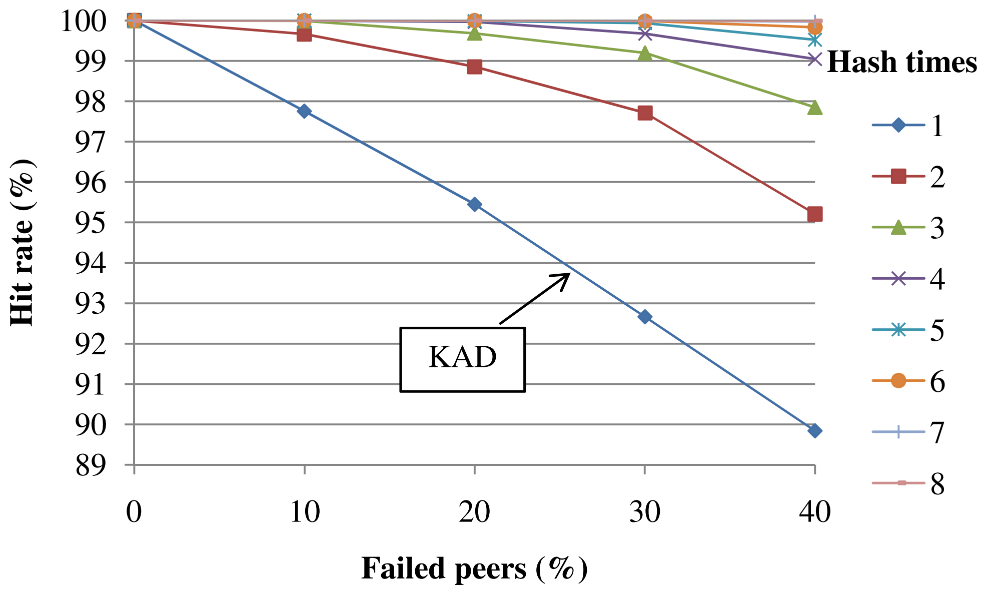


Figure 17. The hit rate with respect to failed peers under different hash times.

Figure 17 shows that the hit rate is close to 100% if we hash more than 5 times with 27% of peers failed. The proposed KAD-N will increase the search resilience in the situation of a large number of peers failed. Because more hash times do not always bring more efficiency, we used a cost-effectiveness factor *k* to determine the maximum hash times.

$$k = \frac{hit\ rate}{total\ network\ traffic * \sigma}$$

To have a larger k, one has to increase the hit rate, and reduce the total network traffic and the standard deviation. From Figure 18, k of 6, 7 and 8 hash times are very close, and k of 7 hash times is the highest. That is, hashing 7 times is the optimal choice for the trace we simulated.

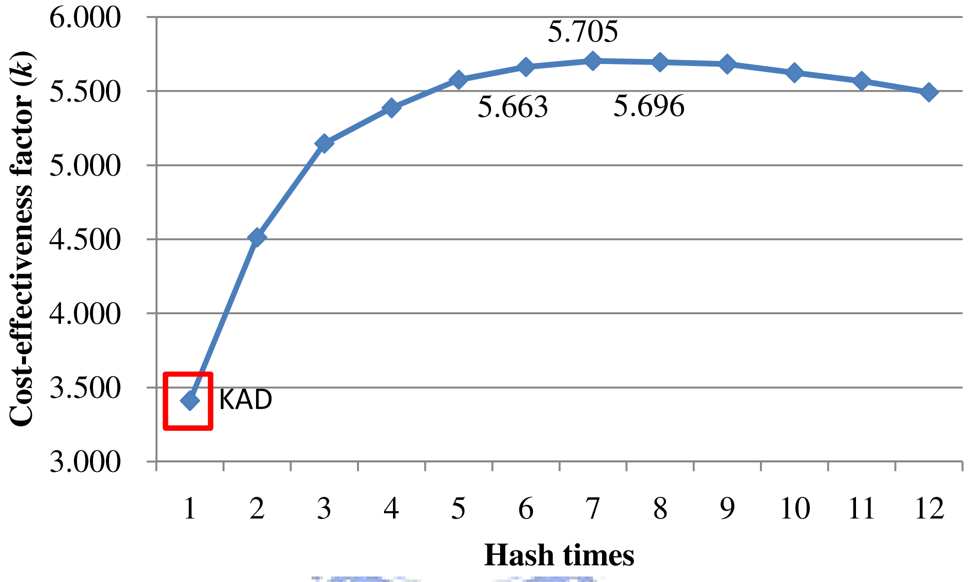


Figure 18. Cost-effectiveness factor under different hash times.

Chapter 5
Implementation Issues

5.1 Applying KAD-N to existing KAD P2P networks

In this chapter, we describe how to implement the proposed KAD-N method. Our method is an improvement of the original KAD. We can implement it based on an existing KAD P2P network, such as eMule [8] or aMule [24]. They are both open source projects so we can get their source codes easily. By modifying their source codes, the proposed KAD-N method can be implemented. In the following, we describe how to adapt a KAD method to the proposed KAD-N method.

Figure 19 shows the publishing procedure of the original KAD P2P network and Figure 20 shows its search procedure. We can implement the proposed KAD-N method based on a KAD P2P network by modifying the hash operation for publishing and searching objects. To apply our method to the KAD P2P network, we replace block B1 in Figure 19 with block A1 in Figure 11 and also replace block B2 in Figure 20 with block A2 in Figure 12.

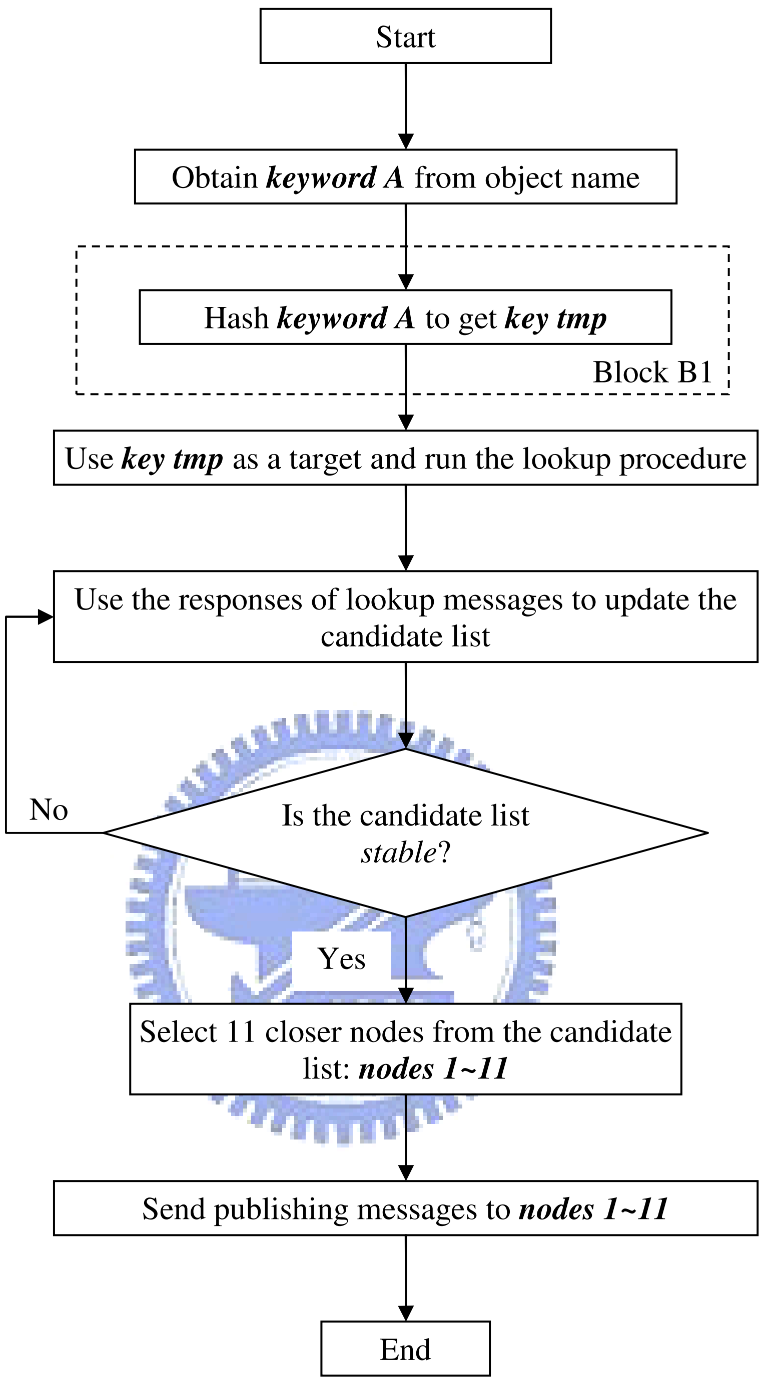

Start
Obtain *keyword A* from object name
Hash *keyword A* to get *key tmp*
Block B1
Use *key tmp* as a target and run the lookup procedure
Use the responses of lookup messages to update the candidate list
Is the candidate list *stable*?
No
Yes
Select 11 closer nodes from the candidate list: *nodes 1~11*
Send publishing messages to *nodes 1~11*
End

Figure 19. The publishing procedure of KAD.

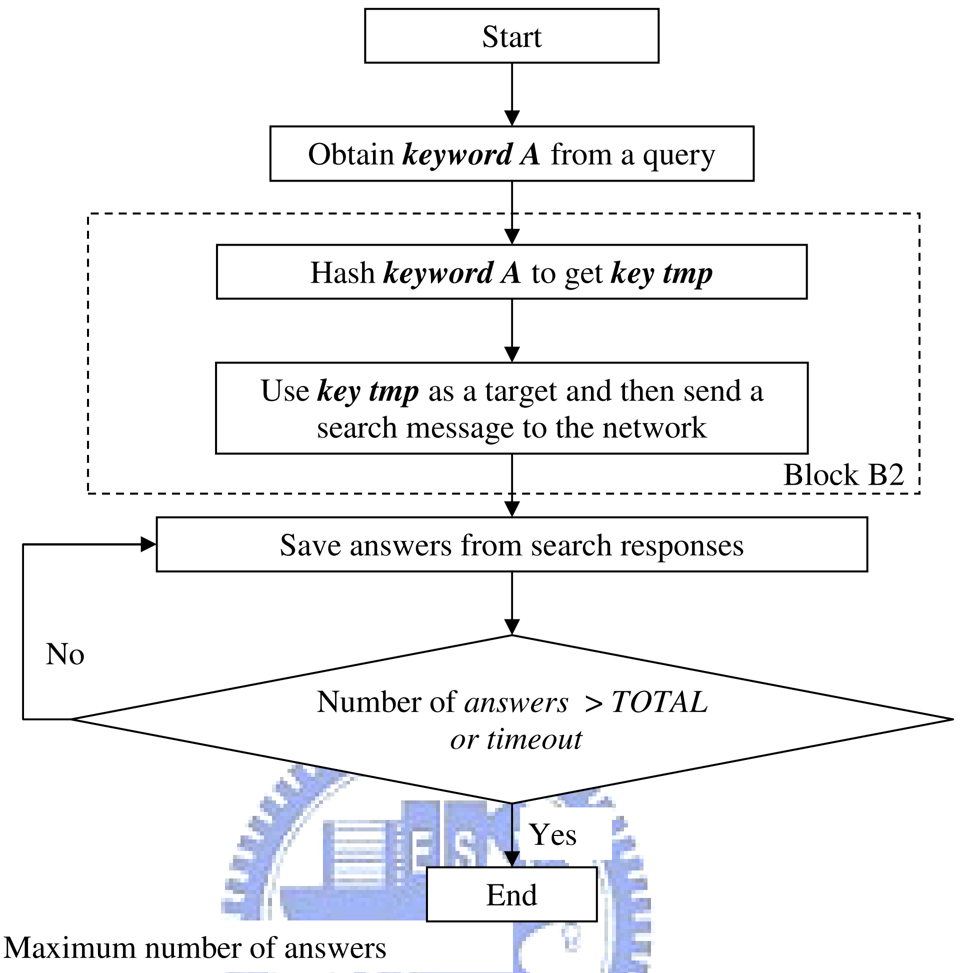

Start
Obtain *keyword A* from a query
Hash *keyword A* to get *key tmp*
Use *key tmp* as a target and then send a search message to the network
Block B2
Save answers from search responses
No
Number of *answers* > *TOTAL* *or timeout*
Yes
End
TOTAL: Maximum number of answers

Figure 20. The search procedure of KAD.

5.2 Combining with streaming

The KAD P2P network can be extended to support streaming applications. Most of existing P2P networks based on KAD are only capable of file sharing. We can enhance the ability of KAD P2P networks by adding some functions, such as P2P streaming. For example, if a video file has been published by several peers in the KAD network, we can use a P2P streaming tool to view this file while downloading. In the original KAD P2P network, we must wait until the whole file is downloaded. It is not efficient.

To combine a KAD P2P networks with streaming, we can modify its download function. A peer can search a video file to get a list of peers who have this file in the KAD P2P network. We can use this peer list to form a P2P streaming network. Peers who have the requested video file will be the sources in the P2P streaming network. Other peers who want to watch this video file can join the P2P streaming network. The KAD-N method we proposed can improve the search hit rate. It may increase the probability of finding more peers who have the requested video file. By applying our method, the P2P streaming network can achieve resilience in case that some peer failed.

Chapter 6 Conclusions

6.1 Concluding remarks

The proposed KAD-N method does balance load of each keyspace and also improve the search hit rate. It is a simple and effective method. By hashing random times when publishing a keyword, indexes can be distributed more even and the publishing load of each peer would be more balanced. Although KAD-N may slightly increase the number of total messages in the KAD network, the extra traffic is very small. Based on the simulation results, the optimal hash times is 7, which improves the hit rate to close to 100% and cause about 7% of extra traffic. Our method can not only improve the search resilience but also balance the publishing load between peers in KAD networks. In addition, the proposed KAD-N method can be extended to support other DHT based P2P networks.

6.2 Future work

Our KAD-N method is a simple and effective way to achieve load balancing and search resilience. There are some issues that deserve to be further studied. (1) Adapt our method to let it be applicable to other DHT based P2P networks. In this thesis, our method is based on KAD P2P networks. Because of the differences of search and publishing mechanisms between KAD and other DHT based P2P networks, our method needs to be adapted for applying to other networks. (2) Support KAD P2P networks with P2P streaming applications. Most of existing networks built by KAD are only capable of file sharing. In Chapter 5, we discussed how to combine a KAD P2P network and our method for P2P streaming. This issue deserves to be further studied as well.

Bibliography

[1] D. Kundur, Z. Liu, M. Merabti, and H. Yu, “Advances in peer-to-peer con tent search,” in *Proceedings of the IEEE International Conference on Multimedia and Expo*, pp. 404-407, July 2007.

[2] Y.J. Joung, L.W. Yang, and C.T. Fang, "Keyword search in DHT-based peer-to-peer networks," *IEEE Journal on Selected Areas in Communications*, vol. 25, pp. 46-61, January 2007.

[3] P. Maymounkov and D. Mazieres, “Kademlia: A peer-to-peer informatiion system based on the XOR metric”, in *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, pp. 53-65, March 2002.

[4] Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for Internet applications,” in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp. 149-160, August 2001.

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A scalable content-addressable network,” in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp. 161-172, August 2001.

[6] Rowstron and P. Druschel, “Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems,” in *Proceedings of the 2001 IFIP/ACM International Conference on Distributed Systems Platforms,* vol. 2218, pp. 329-350, November 2001.

[7] Y. Zhao, J. Kubiatowicz, and A. D. Joseph, “Tapestry: An infrastructure for fault-tolerant wide-area location and routing,” University of California, Berkeley, Tech. Rep. UCB/CSD-01-1141, April 2001.

[8] “eMula Project,” [Online]. Available: http://www.emule.com/.

[9] “BitTorrent,” [Online]. Available: http://www.bittorrent.com/.

[10] “Gnutella website,” [Online]. Available: http://www.gnutella.com.

[11] “Fasttrack peer-to-peer technology,” [Online]. Available: http://www.fasttrack.nu/.

[12] “iMesh website,” [Online]. Available: http://www.imesh.com.

[13] “JXTA community projects,” [Online]. Available: https://jxta.dev.java.net.

[14] M. Abdelaziz, B. Traversat, and E. Pouyoul, "Project JXTA: A loosely-consistent DHT rendezvous walker," March 2003. [Online]. Available: http://www.jxta.org/docs/jxta-dht.pdf.

[15] T. H. Chang, “Keyword search for enhancing JXTA discovery service in peer to peer networks,” *Master’s Thesis*, National Chiao Tung University, June 2008.

[16] M. Steiner, D. Carra, and E. W. Biersack, “Faster content access in KAD,” in *Proceedings of the Eighth International Conference on Peer-to-Peer Computing*, pp. 195-204, September 2008.

[17] D.Wu, Y. Tian, and K.W. Ng, “Achieving resilient and efficient load balancing in DHT-based P2P networks,” in *Proceedings of the 31st IEEE Conference on Local Computer Networks*, pp. 115-122, November, 2006.

[18] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, “Load balancing in dynamic structured p2p systems,” in *Proceedings of the IEEE INFOCOM’04*, pp. 2253-2262, March 2004.

[19] J. Byers, J. Considine, and M. Mitzenmacher, “Simple load balancing for distributed hash tables,” In *Proceedings of the IPTPS’03*, pp. 80-87, October 2003.

[20] R. Brunner, “A performance evaluation of the KAD-protocol,” *Master’s Thesis*, University of Mannheim and Institut Eurecom, November 2006.

[21] M. Steiner, W. Effelsberg, T. En-Najjary, and E. W. Biersack, “Load reduction in the KAD peer-to-peer system,” in *Proceedings of the 5th International Workshop on Databases, Information Systems and Peer-to-Peer Computing*, October 2007.

[22] M. Steiner, T. En-Najjary, and E. W. Biersack, “A global view of KAD” in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, pp. 117-122, October 2007.

[23] E. W. Biersack, “Everything you want to know on KAD,” June 2008. [Online]. Available: http://www.thlab.net/old/rescom2008/talks/E-Biersack_KAD-tut.pdf.

[24] “aMule Project,” [Online]. Available: http://www.amule.org/.

[25] “WiredReach,” [Online]. Available: http://www.wiredreach.com/.

[26] “Collanons Workplace,” [Online]. Available: http://www.collanos.com/.

[27] X. L. Fu and Y. Xu, “A load balance algorithm for hybrid P2P network model,” in *Proceedings of the ISECS International Colloquium on Computing, Communication, Control, and Management*, pp.236-239, August 2008.