國立交通大學

網路工程研究所

碩士論文

高效率非集中式之KAD同儕網路負載平衡策略

An Efficient Decentralized Load Balancing Scheme in KAD Peer-to-Peer Networks

研究生：徐崇驫

指導教授：王國禎 博士

中華民國九十九年六月

高效率非集中式之KAD同儕網路
負載平衡策略

An Efficient Decentralized Load Balancing Scheme
in KAD Peer-to-Peer Networks

研究生：徐崇騵　　　　Student：Chung-Yuan Hsu

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 Computer Science and Engineering

Department of Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Sciencmae

June 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年六月

高效率非集中式之KAD同儕網路
負載平衡策略

學生：徐崇驟　　　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且被詢問負載所困擾，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，我們提出一個以同餘定理為基礎的方法（KAD-mod）以平衡網路中各節點的負載，我們給予每個節點一個同餘編號，並且設定一個門檻（*RFT*）用以限制一個節點可以負擔的相同索引數。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，*RFT* = 6000 是最佳的設定。我們利用基尼係數來評估相關的負載平衡方法，在基尼係數中0為最平衡情

況，1為最不平衡情況。由模擬結果得知，KAD-mod可以提升搜尋命中率至98.24%，在發佈負載平衡方面，KAD-mod， KAD-7及KAD 在基尼係數的表現上分別為0.23，0.80，0.93。而詢問負載平衡方面，KAD-mod， KAD-7及KAD在基尼係數的表現上分別為0.33，0.67，0.83。評估結果顯示本方法的發佈及詢問負載皆比KAD及KAD-7更平衡。但它會增加8%的額外流量及平均需要額外的0.5 hop才可找到發布目標節點。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，本方法可以很容易地被延伸應用至其他以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。

An Efficient Decentralized Load Balancing Scheme in KAD Peer-to-Peer Networks

Student: Chung-Yuan Hsu Advisor: Dr. Kuochen Wang

Department of Computer Science
National Chiao Tung University

Abstract

Kademlia (KAD) peer-to-peer (P2P) networks have become more and more popular. However, they have an unbalanced publish load problem. It causes a few peers to store a large number of indexes and these peers will become hotspots. Once become a hotspot, the peer must handle a large number of requests that result in high load, and it become a network bottleneck. To conquer this problem, we propose a modulo based method (called *KAD-mod*) to balance load in the KAD network. We give each peer a new ID (called *mod ID*) using modular arithmetic. A *request forwarding threshold* (*RFT*) is used to help decide if an index should be redirected to the same mod ID of peers in another zones. This method allows the same mod ID peers to share load. We used *Gini coefficient* (G, $0 \leq G \leq 1$, 0: fully balanced) as a load balancing index to evaluate representative load balancing methods. Simulation results show that the proposed *KAD-mod* has the search hit rate close to 100%. The G's of KAD-mod, KAD-7, and KAD for publishing load are 0.23, 0.80, and 0.93, respedtively. As to G for request load, KAD-mod is 0.33, KAD-7 is 0.67, and KAD is 0.83. We can see that the proposed KAD-mod is much more load balancing than the other methods. However, KAD-mod has 8% extra traffic and the hop count per publish increases from 2.5 to 2.9. By enhancing the search hit rate, KAD-mod can improve the search resilience of KAD P2P networks with failed peers. Furthermore, the proposed KAD-mod method can be easily extended to other DHT-based P2P networks.

Keywords: KAD , load balancing, , 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-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 peer-to-peer (P2P) application is one of the most important applications in the internet. The most popular P2P based applications are file sharing systems, storage systems and communication systems. P2P accounts for more than 73% of the internet traffic at the end of 2007 [27]. Moreover, there are millions of simultaneous connected P2P users spread out on different continents and states [16].

There are three main types of P2P overlays: unstructured P2P overlays, hybrid P2P overlays, and structured P2P overlays. Structured overlays account for 99% of all the P2P network traffic and KAD, the most popular structured P2P overlays account for 95% of the structured P2P traffic [27]. We will explain why structured P2P overlays (networks) are the most popular P2P networks.

1.1 Why structured P2P networks

Structured P2P networks [5, 6, 7, 8, 9] are based on distributed hash table (DHT). They were developed to improve the performance of data discovery. They impose constraints both on the node graph and on data placement to enable efficient discovery of data [29]. When a peer wants to share an object, it needs to decide which peer the index should be stored. Using a hashing function can achieve the purpose that maps an object’s name to a unique peer in the network. If a peer wants to find an object, it first hashes the object’s name to get a peer and then queries the peer to get an index. Using the index, it can find the actural peers that store this object. These structured P2P networks differ basically in how peers maintain their routing tables to guarantee an efficient route between peers. KAD [5] is a Kademlia-base P2P DHT based routing protocol implemented by several applications such as eMule [10], BitTorrent [11] and aMule [12].

Unstructured P2P networks, such as Gnutella [2], Napster [3], Freenet [4], organize peers into a random graph and use floods or random walks to discover data stored by overlay peers. This approach supports arbitrarily complex queries and it does not impose any constraints on the peer graph or on data placement; for example, each peer can choose any other peer as its neighbor in the overlay [29]. Unstructured P2P networks often uses TTL to restrict discovery time and discovery scope. They cannot find rare data items efficiently because it requires to visit a large fraction of overlay peers. It may causes a lot of network traffic.

Hybrid P2P networks combine the flooding and DHT. To improve search quality and efficiency for both popular and rare items, the hybrid P2P network is introduced [13, 14]. By combining the above two architectures, queries are handled in a hybrid manger: popular objects are found via flooding, while rare objects are found via a DHT-based algorithm.

Table I shows that the structured P2P network is an effective design for file sharing. The structured P2P network can guarantee to search not only popular objects but also rare objects, and it causes network traffic much lower than the unstructured P2P network. The hybrid P2P network's searching scheme depends on how popular the object is. However, defining popular objects is a complex problem. So compared with hybrid P2P networks, the search mechanism in structured P2P networks is much simpler. Therefore, in this thesis, we will focus on structured P2P networks. In structured P2P networks, we chose KAD since it is the most popular structured P2P networks [27].

Table I. Comparison of three main P2P architectures

Architecture	Unstructured P2P	Structured P2P	Hybrid P2P
Routing scheme	Flooding	DHT	Flooding and DHT
Guaranteed search	No	Yes	Yes
Network traffic	High	Low	Middle
Index is stored in	Local peer	Foreign peer	Local or foreign peer
Search efficiency	*O(N)*	*O(logN)*	*O(logN)*

1.2 Load balancing problems in structured P2P networks

Many solutions have been proposed to solve load balancing problems in structured P2P systems. Generally speaking, there are publish, request, and routing load balancing problems in structured P2P networks. The publish load balancing problem is the most important problem since it will result in request and routing load balancing problems. To resolve the publish load problem, most of the solutions reassign loads from heavy loaded peers to light loaded peers. There are three difficult issues, where the loads be reassigned, where to find the reassigned loads, and how to achieve a better tradeoff to resolve the above two issues.

1.3 Motivation

In this thesis, we want to balance the publishing load in KAD P2P networks as much as possible to avoid peers becoming hotspots. In structured P2P systems, each data item is mapped to a unique identifier (ID) and the peer with this ID stores all the indexes that are mapped into it. Structured P2P systems could result in an $O(\log N)$ imbalance factor in the number of objects stored at a peer, where N is the number of peers in the system [28]. Heavy load peers may become hotspots and cause network congestion around them and also affect routing performance. Once these peers become offline, the search hit rate will decrease dramatically. Load imbalancing is a critical problem that must be treated property in order to fairly use available physical resources.

1.4 Problem statement

In this thesis, we intend to resolve two problems in KAD P2P networks. The first one is the publish load imbalancing problem and the other is the search hit rate problem. By the structured P2P network mechanism, each keyword produces a key. Each key has a specific target peer to publish. Target peers use these keys to construct indexes of objects. If a keyword is popular, the target peer will handle unusually large indexes. In this way, a few peers may handle most of the indexes. It may cause unbalanced loads between peers. As to the search hit rate, because peers are online or offline frequently in KAD P2P networks, peers offline or failure may result in indexes lost. Although indexes may be lost, actually, the related objects still exist in the network. To find these objects, it will lower the search hit rate.

1.5 Thesis organization

The rest of this thesis is organized as follows. Chapter 2 reviews the background of KAD P2P networks and existing load balancing schemes in structured P2P networks. In Chapter 3, we present our design approach in detail. Simulation setup and simulation results are presented in Chapter 4 and Chapter 5 concludes the thesis and outline future work.

Chapter 2

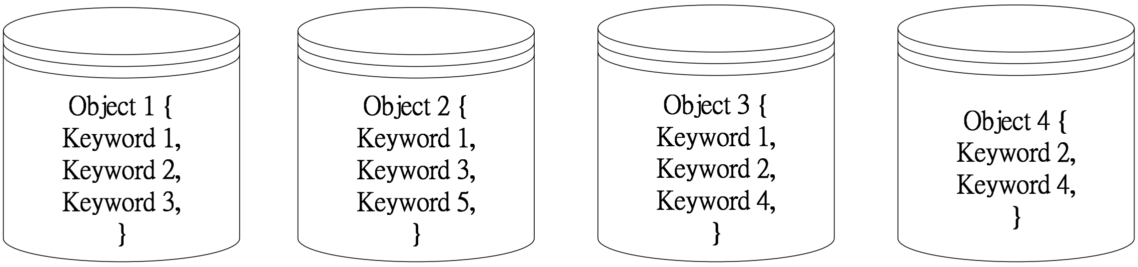
Preliminaries and Related Work

Since the KAD P2P network is our main target to enhance, in this chapter, we review the KAD P2P network and some existing load balancing methods. KAD P2P networks are based on distributed hash table (DHT). First we review the DHT and then describe the mechanism of how to lookup, publish, and search objects in the KAD P2P network. Finally, we review existing load balancing mechanisms, KAD [5], KAD-7 [23], and MHF [24].

2.1 Distributed hash table

DHT is based on a consistent hashing function. In DHT-based P2P networks, each object is assigned a unique ID using a consistent hashing function. A DHT-based P2P network basically supports 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 network to enhance search functionality. The most common way to implement keyword search in information systems is by inverted index [15]. 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.

A distributed inverted index is built to implement keyword search in a structured P2P network. By using DHT-based P2P 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 [15]. In Figure 1(a), we show an inverted index example of four objects: Objects 1, 2, 3, and 4. Object 1 contains Keywords 1, 2, and 3, and Object 2 has Keywords 1, 3, and 5, etc. Figure 1(b) shows the inverted index which is built by these keywords and objects. Following this rule, we can link keywords to different objects. For example, if we use “Keyword 2” as a keyword to search the network, we can find Objects 1, 3, and 4.



(a) Each object has some keywords

Keyword	Object set
Keyword 1	{Object 1, Object 2, Object 3 }
Keyword 2	{ Object 1, Object 3, Object 4}
Keyword 3	{ Object 1, Object 2}
Keyword 4	{ Object 3, Object 4}
Keyword 5	{ Object 2}

(b) An inverted index

Figure 1. An inverted index example of four objects [15].

2.2 Background of KAD

Each KAD node has a global identifier, referred as KAD ID, which is 128-bit long and randomly generated by a cryptographic hash function. The designers of KAD decided to consider a contact sufficiently close to the target if it shares with it at least the first 8 bits. The space of KAD IDs that satisfy this constraint is called tolerance zone [17]. There are $2^8 = 256$ zones in a KAD P2P network. We will briefly explain the lookup, publishing, and searching procedures in KAD P2P networks.

2.2.1 Lookup procedure

When searching for some objects, a 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. Routing in KAD is based on prefix matching. In KAD networks, the distance between two nodes is calculated by *XOR-distance*. The XOR-distance is defined as $d(a, b) = a \oplus b$. It calculated bitwise on the KAD IDs of two nodes, e.g., the distance between $a = 10011$ and $b = 01111$ is $d(a, b) = 10011 \oplus 01111 = 10100$. Routing to a KAD ID is done in an iterative way. Figure 2 is an example lookup procedure. In the first step, the searching peer has three closest possible contacts from the routing table. They have different XOR-distances and are still not close enough to the target peer. The second step in Figure 2 shows that the searching peer received three responses. The searching peer obtains three more closer possible contacts by the responses. If a new possible peer in the tolerance zone, it will be stored to a list called the candidate list. In the third step, two of these possible peers are in the tolerance zone. These two peers will be saved to the candidate list. In the fourth step, the searching peer sends a request for more closer peers to the three closest peers again. 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 three candidate peers [17]. At this point, no new request is sent and the candidate list becomes stable. KAD travels only $O(logN)$ peers during the execution of the lookup procedure when there are N peers in the network.

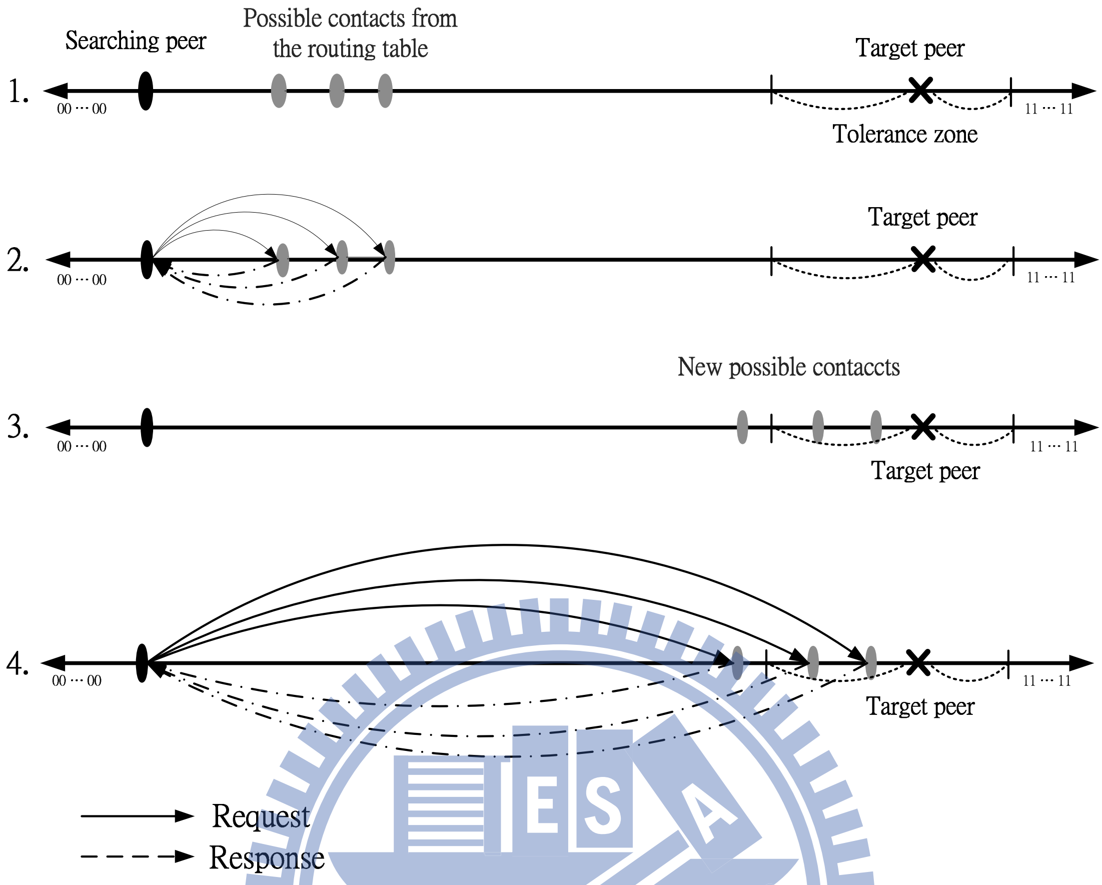


Figure 2. An example iterative lookup procedure [16].

2.2.2 Publish procedure

Publish is an essential action when peers want to share objects. Peers will publish keyword keys and a source key to foreign peers. In Figure 3, the KAD ID of the peer is "10111." An object can produce two different keys, a source key and keyword keys. A source key is computed by hashing the name of the object. Keyword keys are computed by hashing keywords from the name of the object [16]. The keywords of this object are "Modular" and "KAD." In Figure 3, the source key is "01011" and the keyword keys of "Modular" and "KAD" are "00001" and "00100," respectively

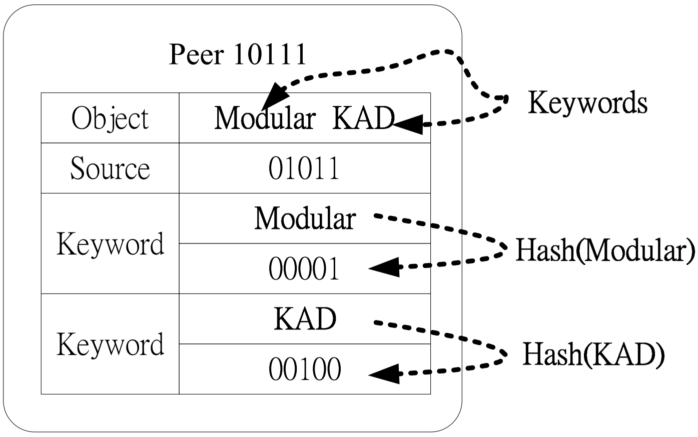


Figure 3. An example of an object to be published.

Figure 4 shows an example of publishing steps for an index. Before publishing an index, a sending peer must use KAD_REQ to find a receiving peer. At first the sending peer sends a KAD_REQ to the receiving peer. KAD_REQ is used to find the receiving peer and check whether the peer is alive. When the receiving peer receive KAD_REQ, it will send a KAD_RES back. After establishing a connection between the sending peer and the receiving peer, the sending peer starts to publish keys to the receiving peer.

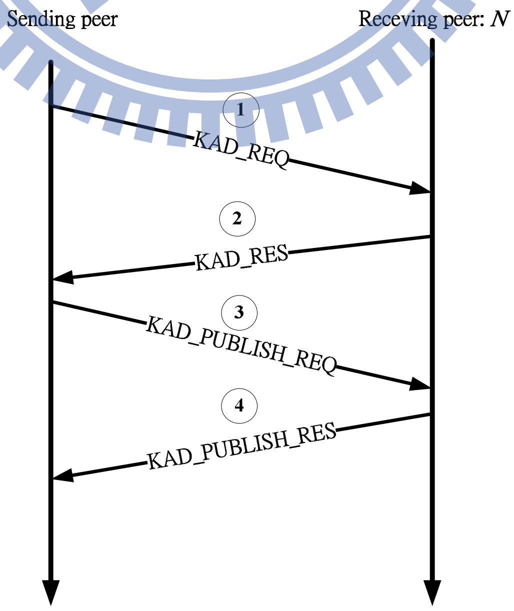


Figure 4. The KAD publish steps for an index [16].

When a peer starts to publish keys, the peer will publish a source key and keyword keys by 2-level publishing scheme. Figure 5 shows an example 2-level publish. A peer “10111” wants to publish an object named “Modular KAD.” This object name will result in two keywords, “Modular” and “KAD.” All relevant references to the original object are generated, such as the source key and the keyword keys. Next, keyword keys “Modular 00001” and “KAD 00100” are published to corresponding peers “00001” and “00100” to build indexes, which are all pointed to peer “01011.” Finally, the source key is published, with an index pointing to the publishing peer.

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 [17].

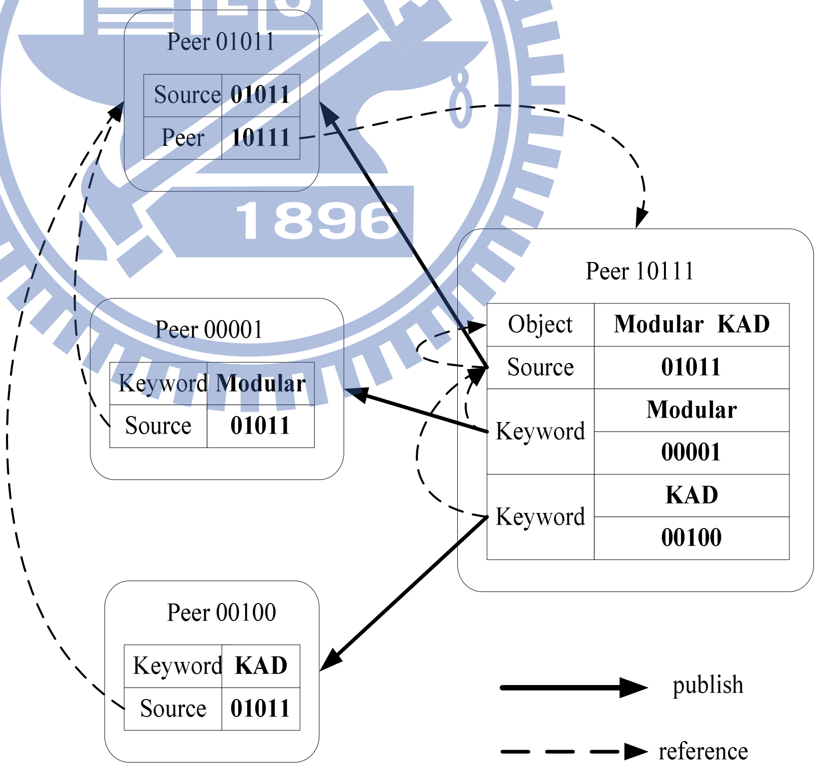


Figure 5. An example of 2-level publish.

2.2.3 Search procedure

Like publishing, searching files is also a 2-level search: keyword search and source search. For a keyword search, the hash value of the first word of the user input is computed. The rest of words are packed in a form of a search tree. A query consists of a hash value of the first keyword and a search tree [16]. The query is routed to the peers that have a KAD ID close to the hash value. The matching results are responded from that peers and carry the information of source keys. For a source search, a user chooses a desired object from returned results. Then the source key of the object is used for searching the peers who have the object. The returned results would be added to the download queue of the object.

2.3 Original load balancing scheme in KAD

KAD limits the number of indexes in each peer to avoid overloading. A peer can handle a maximum of 60,000 indexes and can hold a maximum of 50,000 indexes of an individual keyword. Therefore, when a peer reaching the limit of maximum indexes number receives a publishing request, it will reply a successful message, even if the publishing request is rejected.

2.4 Other existing load balancing schemes

KAD-7 [23] hashes the keyword of an object r times to produce a key for publishing objects, where r is a random number and $1 \leq r \leq 7$. Different peers may hash different times to produce different indexes, which all represent the same keyword. These keys will be published to different peers, not just to one peer. Because indexes are spread, KAD-7 will increase the number of search messages.

In [18], the authors found that the peaks of load are due to very popular keywords that are most often meaningless stopwords. They proposed to add a stopword exclusion step into all KAD based P2P systems. They use stopword exclusion to reduce the number of indexes, so the total indexes in the KAD will decrease.

In [24], the authors describe a novel approach with multiple hash functions (MHF) to replicate the hotspots in a series of different nodes to distribute the high load evenly, and it can increase or decrease the replicas dynamically. MHF provides a load balancing scheme for a high request rate. If the request rate is not over a threshold, the zone with popular keywords will still have a large number of requests. MHF will result in a lot of additional network traffic because a large number of indexes are replicated.

2.5 Qualitative comparison of representative load balancing schemes

Table II shows the comparison of four representative load balancing schemes. In the proposed KAD-mod, the publish load and request load are more balanced and thus the search hit rate will increase. If a popular keyword references 10^6 indexes, KAD-mod can distributes indexes more even to 160 zones. In contrast, KAD-7 can only spread indexes to seven zones and KAD only to just one zone. Since KAD-mod and KAD-7 will spread the indexes, there will be more peers that have the same indexes. As a result, the search hit rates of both schemes will increase in case that some peers failed. However, their network traffic will increase slightly because of increased search messages. MHF uses a scheme that replicates the hotspot load to other peers, so it will generate a lot of indexes which increase the network traffic. Peers to know where the key is located must ask the original responsible peer. Once the original responsible peer becomes offline, the search hit rate will decrease.

Table II. Qualitative comparison of four load balance schemes.

Approach	KAD [5]	MHF [24]	KAD-7 [23]	KAD-mod (proposed)
Publishing load	High	High	Middle	Low
Request load	High	Middle	Middle	Low
Search hit rate	Middle	Middle	High	High
Network traffic	Middle	High	Middle	Middle

Chapter 3

Proposed KAD-mod Load Balancing Scheme

KAD has a 128-bit ID space and there are 256 zones in a KAD P2P network. Divide 2^{128} by 256 to get a quotient of 2^{120} so that each zone has at most 2^{120} peers. We will define a new ID type: *mod ID*. A mod ID is computed as KAD ID mod 2^{120}. By deriving a new mod ID (as follows), for each peer, the peers with the same mod ID will be located in different zones. In Figure 6, assume that there are 15 peers and 5 zones so that each zone has 3 peers. Each peer's KAD ID mod 3 will generate its mod ID. For example, 4 ≡ 1 mod 3 and "1" is the mod ID. For example, a peer's KAD ID is *N*, then peers with KAD ID $N+2^{120}$, $N+2\times2^{120}$, …and $N+255\times2^{120}$ will all have the same mod ID.

Deriving a new mod ID:

Let *n* be a nonzero integer. We say that two integers *a* and *b* are *congruent modulo n* if there is an integer *k* such that $a - b = kn$. In the KAD case, we have $a \equiv b \bmod n$, where a = KAD ID, $n = 2^{120}$, b = mod ID.

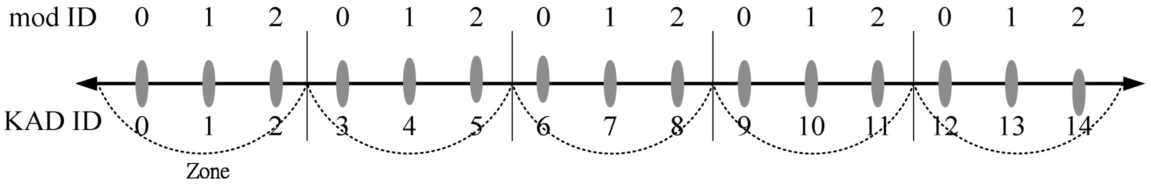


Figure 6. A mapping between KAD IDs and mod IDs in the KAD P2P network.

3.1 Concept of KAD-mod

We propose a modulo based load balancing method to let peers with the same mod ID share loads. Using mod ID, we can easily find where loads are reassigned and where to find the reassigned loads. We use a *request forwarding threshold* (*RFT*) to help decide if an index should be redirected to the same mod ID of a peer at another zone. We limit the number of indexes stored in each peer to avoid overloading. A peer can handle at most *RFT* indexes of an individual keyword. For example, when a peer reaches the limit of *RFT* indexes, it will redirect the remaining requests to the same mod ID of peers at another zones. Figure 7 shows the concept of KAD-mod for publishing. In this example, there are 180 keys *K* published from many peers to peer *N* and *RFT* = 60. Peer *N* will handle the first 60 keys. For the remaining keys, peer *N* redirects 61th to 120th keys to peer $N + 2^{120}$ and redirects 121th to 180th keys to peer $N + 2 * 2^{120}$.

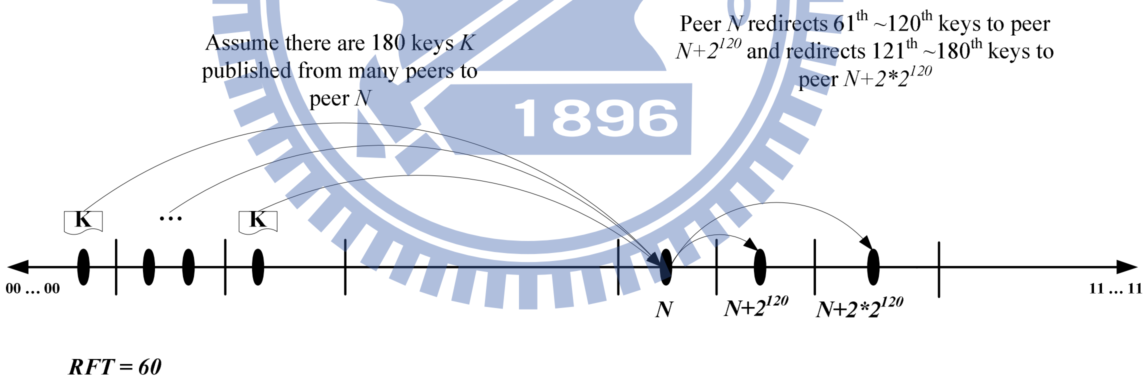

Assume there are 180 keys *K* published from many peers to peer *N*
Peer *N* redirects 61th ~120th keys to peer $N+2^{120}$ and redirects 121th ~180th keys to peer $N+2*2^{120}$
K
…
K
00 … 00
11 … 11
N
$N+2^{120}$
$N+2*2^{120}$
RFT = 60

Figure 7. The concept of KAD-mod for publishing.

3.2 Publish procedure

In KAD P2P networks, a key could be published to a peer from many peers. Each peer has a counter (*REQ_counter*) to record the number of KAD_REQ's for storing the same key. The basic idea is when the number of *REQ_counter* exceeds *RFT*, then the peer will redirect the remaining KAD_REQ's for storing the same key to other zones. In other words, the receiving peer will become a redirection peer when the number of KAD_REQ’s received for storing the same key exceeds *RFT*. Figure 8(a) shows a scenario that the number of KAD_REQ’s peer *N* received for storing the same key does not exceed *RFT*. Figure 8(b) shows an alternative publishing procedure in five steps when the number of KAD_REQ’s peer *N* received for storing the same key *K* is over *RFT*. The five steps are:

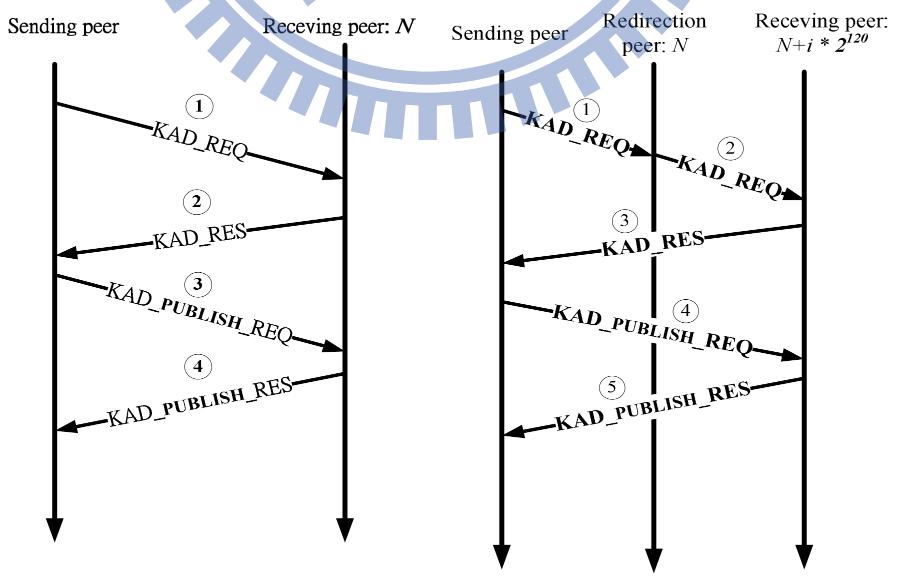
Step 1: When a sending peer wants to publish key *K* to receiving peer *N*, it sends a KAD_REQ to receiving peer *N*.

Step 2: Divide *REQ_counter* by *RFT* to get a quotient *i*. In this case, $i \geq 1$. Receving peer *N* will become a redirection peer and redirect KAD_REQ to peer $N+i\times2^{120}$.

Step 3: When receiving peer $N+i\times2^{120}$ receives KAD_REQ, it will sends a KAD_RES to the sending peer.

Step 4: Then, the sending peer starts to send KAD_PUBLISH_REQ to new receiving peer $N+i\times2^{120}$.

Step 5: When receiving peer $N+i\times2^{120}$ receives KAD_PUBLISH_REQ, it sends KAD_ PUBLISH _RES to sending peer N. Then, keyword *K* is published successfully.



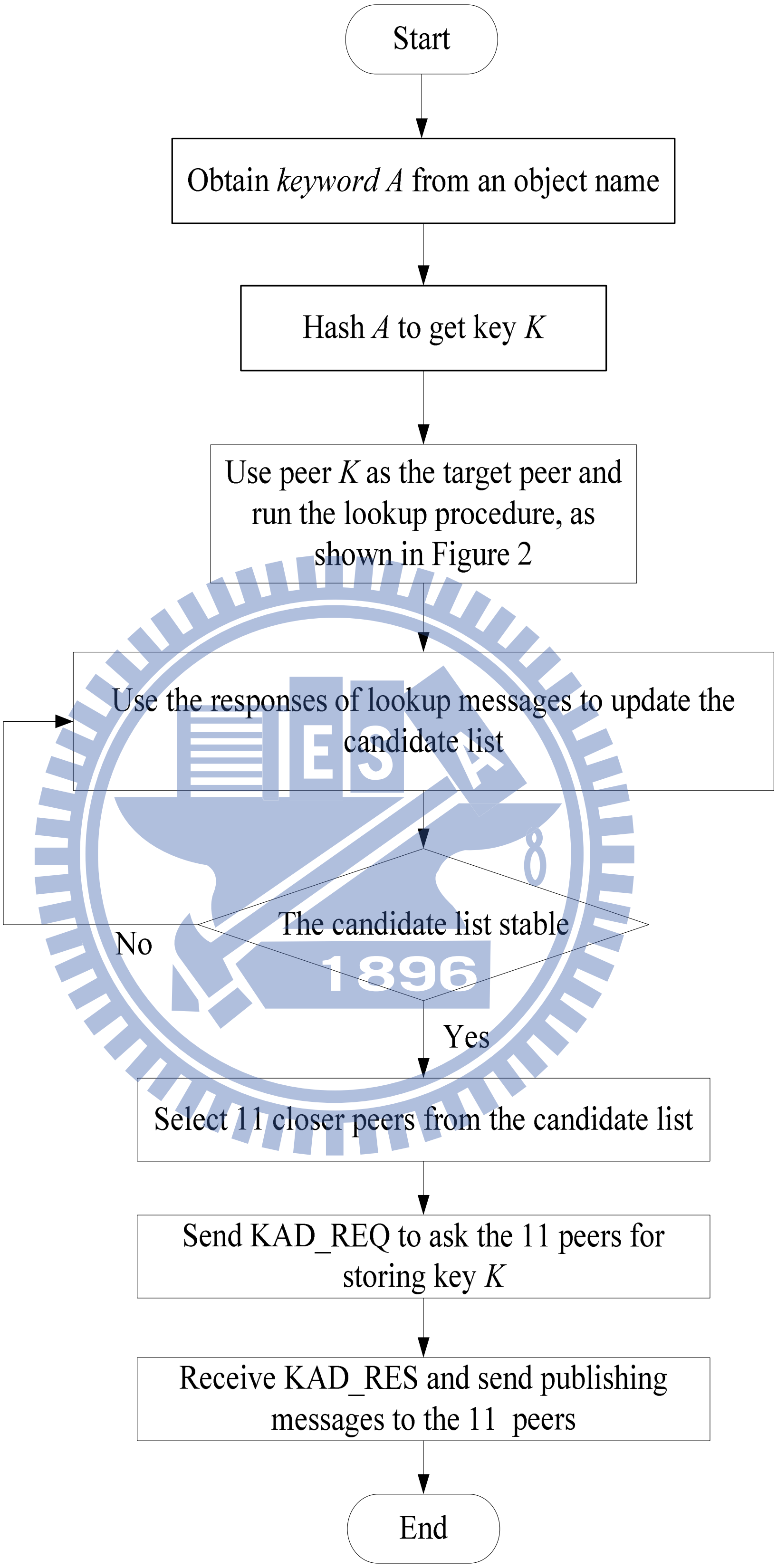
(a) The publish procedure when receiving peer is not overloaded.

(b) The publish procedure when receiving peer is overloaded.

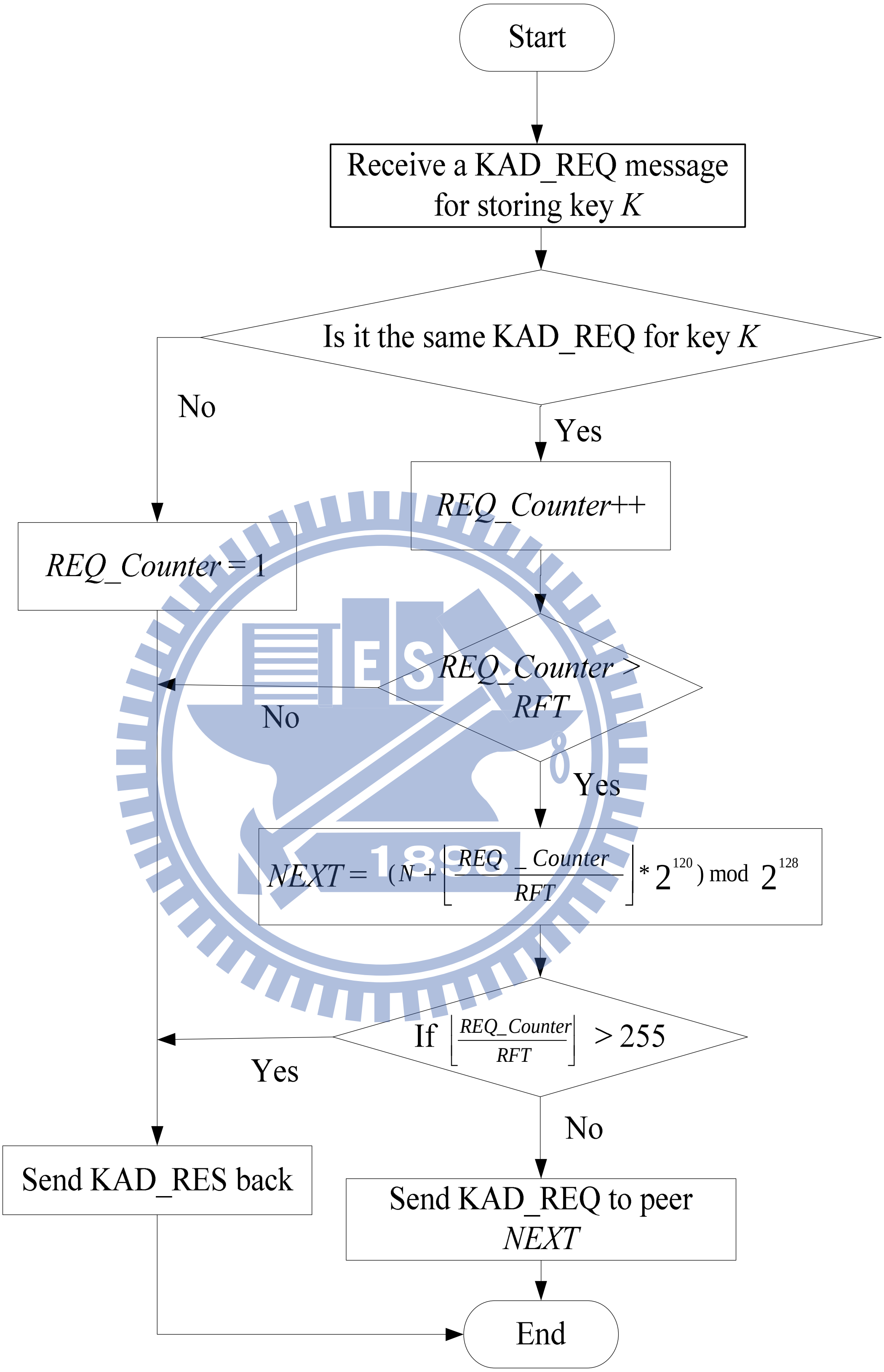
Figure 8. The KAD-mod publish procedure for a key.

The detail of the publishing procedure in KAD-mod is shown in Figure 9. Figure 9(a) shows the procedure that a peer publishes a key. We hash keyword *A* to generate a key *K*. Peer *K* will be the target peer and then the peer uses a lookup procedure to send KAD_REQ to the target peer. The details of the lookup procedure has been presented in Chapter 2. Then we will receive several responses which contain some possible peers who are closer to the target peer. We use these peers to update the candidate list. If the candidate list becomes stable, then go to the the next step. Top 11 peers will be selected from the candidate list for sending KAD_REQ to ask for storing the key. After receiving KAD_RES, the peer sends publishing messages to the 11 peers to complete the procedure of publishing a keyword to the KAD P2P network.

Figure 9 (b) shows the condition that a peer receives KAD_REQ for storing a key *K*. First, the peer will check if it has ever received the same KAD_REQ. If the peer has not received the request before, it initializes a new counter, *REQ_counter*, to *1*. Otherwise, it adds one to *REQ_counter* and the peer checks whether *REQ_counter* > *RFT*. If yes, the peer will calculate a peer number *NEXT* and redirect KAD_REQ to peer *NEXT* with the same mod ID at other zones. If no, the peer will send KAD_RES to the sending peer. In order to avoid an infinite loop, the peer will redirect to at most 255 different zones except its own zone.


Start
Obtain *keyword A* from an object name
Hash *A* to get key *K*
Use peer *K* as the target peer and run the lookup procedure, as shown in Figure 2
Use the responses of lookup messages to update the candidate list
The candidate list stable
No
Yes
Select 11 closer peers from the candidate list
Send KAD_REQ to ask the 11 peers for storing key *K*
Receive KAD_RES and send publishing messages to the 11 peers
End

(a) The procedure of a peer publishing a key.

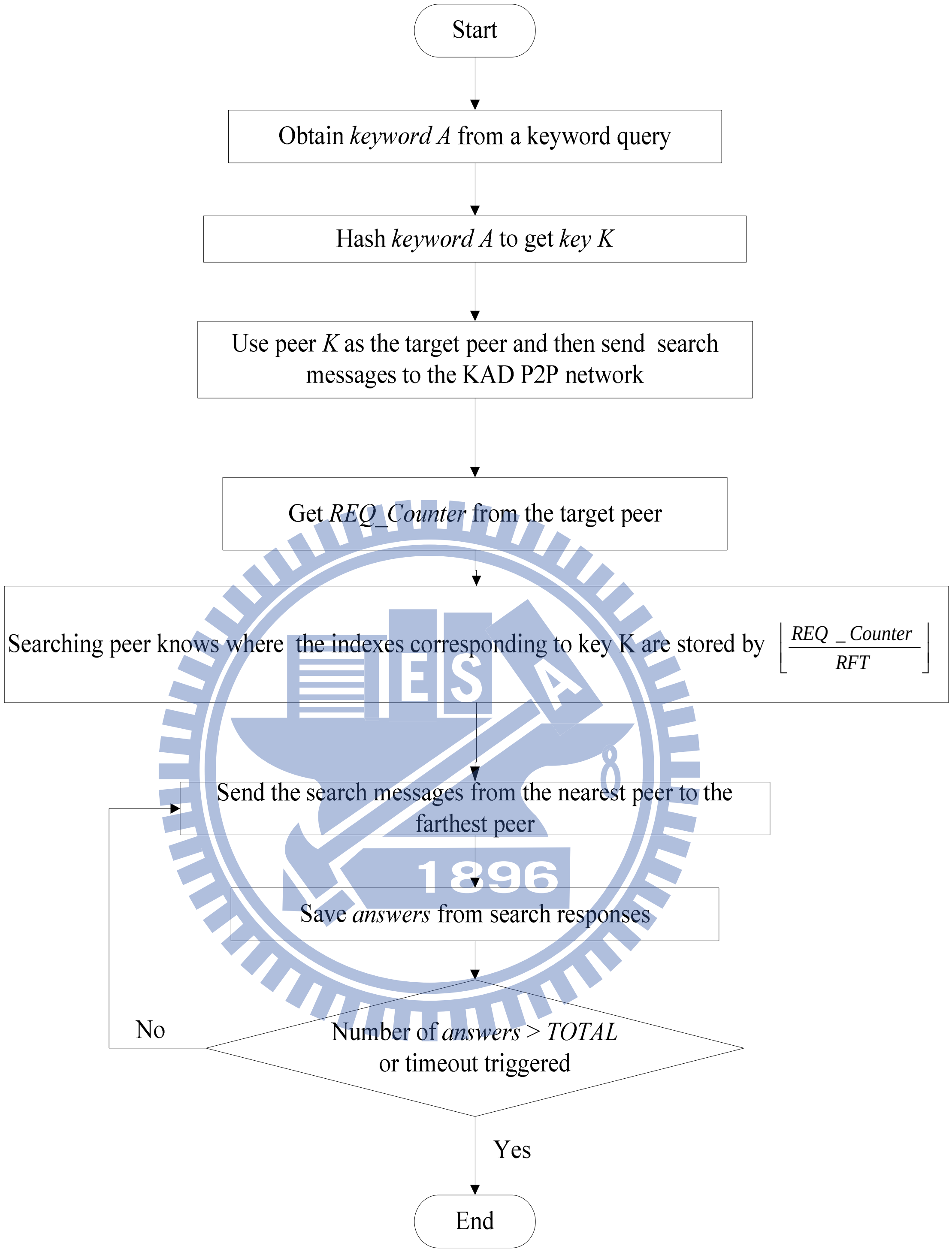

Start
Receive a KAD_REQ message for storing key K
Is it the same KAD_REQ for key K
No
Yes
$REQ\_Counter{+}{+}$
$REQ\_Counter = 1$
$REQ\_Counter > RFT$
No
Yes
$NEXT = (N + \left\lfloor \frac{REQ\_Counter}{RFT} \right\rfloor * 2^{120}) \bmod 2^{128}$
If $\left\lfloor \frac{REQ\_Counter}{RFT} \right\rfloor > 255$
Yes
No
Send KAD_RES back
Send KAD_REQ to peer $NEXT$
End

(b)The procedure of a peer receiving KAD_REQ for storing a key.

Figure 9. The publish procedure in KAD-mod.

3.3 Search procedure

Figure 10 describes the search procedure of KAD-mod. The searching peer obtains a keyword from a keyword query (for example, keyword A). Then, this keyword will be hashed to produce a key *K*. The searching peer uses key *K* as target peer ID to send search messages. The searching peer can know *REQ_counter* by the received response and the searching peer can know where the indexes corresponding to key *K* are stored by *REQ_counter*. After that, we send search messages to these peers from the nearest peer to the farthest peer in our routing table. Then, we will receive several search responses which may contain search answers. The search will stop when a maximum numbers answers, *TOTAL*, has been received or a 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 the searching peer receives more than 300 answers [20].


Start
Obtain *keyword A* from a keyword query
Hash *keyword A* to get *key K*
Use peer *K* as the target peer and then send search messages to the KAD P2P network
Get *REQ_Counter* from the target peer
Searching peer knows where the indexes corresponding to key K are stored by $\left\lfloor \frac{REQ\_Counter}{RFT} \right\rfloor$
Send the search messages from the nearest peer to the farthest peer
Save *answers* from search responses
Number of *answers* > *TOTAL* or timeout triggered
No
Yes
End

TOTAL: Maximum number of answers

Figure 10. The search procedure in KAD-mod.

Chapter 4

Simulation Results

4.1 Simulation setup

First, we analyze the KAD P2P network environment. In [21], the authors crawled a representative subnet of KAD every five minutes for six months. They found that in average, there are 8000 peers in a zone. In [19], the authors spied on one zone in the KAD P2P network for 12 hours. They observed that the number of search messages is 561,542 and the size of search messages is 10.8 MB, while the number of publishing messages is 5,549,183 and the size of publishing messages is 996MB. According to the observation in [19], the average size is 0.019 KB for a search message and 0.18 KB for a publishing message. We classify keywords into *ranks* according to the number of times a keyword appeared. The n^{th} popular keyword is classified as rank *n*. The number of indexes for the n^{th} popular keyword is proportional to $k \times 1/n^{1.63}$ where *k* is the number of indexes for the most popular keyword [18]. According to [18], *k* is about 10^7. The parameter settings of our simulation environment are shown in Table III. We used JAVA to construct our simulation environment. In the simulation, the number of indexes handled by each zone and the number of times each zone being requested were collected for comparison and evaluation.

Table III. Simulation parameter settings.

Number of KAD peers	256 × 8,000
Number of KAD zones	256
Peers per zone	8,000
Number of different keywords	1,000,000
Keywords popularity distribution	Zipf's law [18]
Search distribution	Zipf's law [18]
Raito of publish messages to search messages	10 : 1

4.2 Simulation results

We used *Gini coefficient* (G) as a load balancing index for evaluation of load balancing regarding the number of indexes handled by each zone. The range of G is between 0 and 1. The closer the G approach to 0, the more load balancing it is. G is computed as follows [22]:

$$G = \frac{1}{2\mu}\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\left|l_i - l_j\right| \qquad (1).$$

For calculating G regarding the number of published indexes in each zone. N is the number of zones (N = 256), l_i and l_j are the numbers of indexes handled by the i^{th} and j^{th} zones, respectively, and μ is the average number of indexes handled by each zone. For calculating G regarding the number of requested indexes in each zone, N is the number of zones (N = 256), l_i and l_j are the number of requested indexes in the i^{th} and j^{th} zones, respectively, and μ is the average number of requested indexes in each zone.

Because *RFT* would affect the performance of KAD, KAD-7, and the proposed KAD-mod, we conducted experiments to decide the best *RFT*. Figure 11 shows the *G* regarding the number of indexes published in each zone under a different *RFT*. We found that the lowest value of *G* occurs when the values of RFT are between 5000 and 6000.

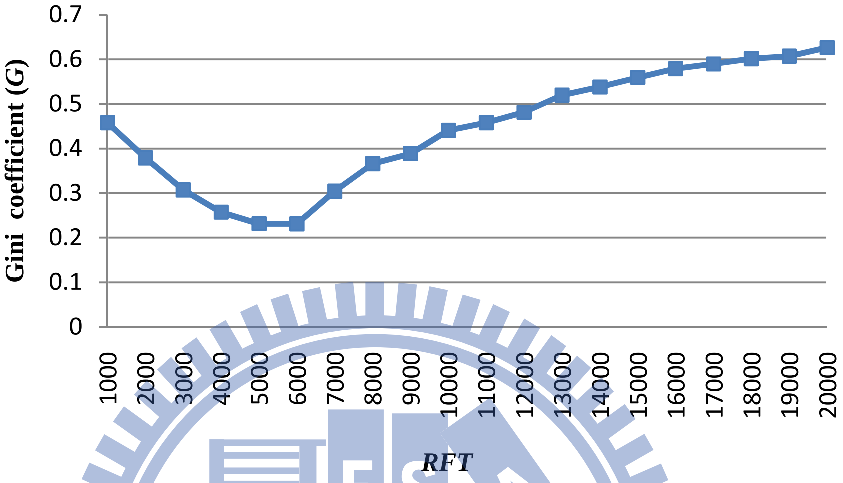


Figure 11. The Gini coefficient regarding the number of indexes published in each zone under a different *RFT*.

There are two issues in the proposed KAD-mod. First, the average hop count of finding a target to publish an index will increase after applying the KAD-mod method. We used the results of [17] to evaluate the average hop count of finding a target to publish an index. Figure 12 shows the average hop count of finding a target to publish an index under a different *RFT*. In our method, for some popular keywords receiving peers may need to redirect KAD_REQs to other peers because the total number of indexes of a popular keyword in the receiving peers exceeds *RFT*. The redirection of KAD_REQs needs an additional hop to find the next target.

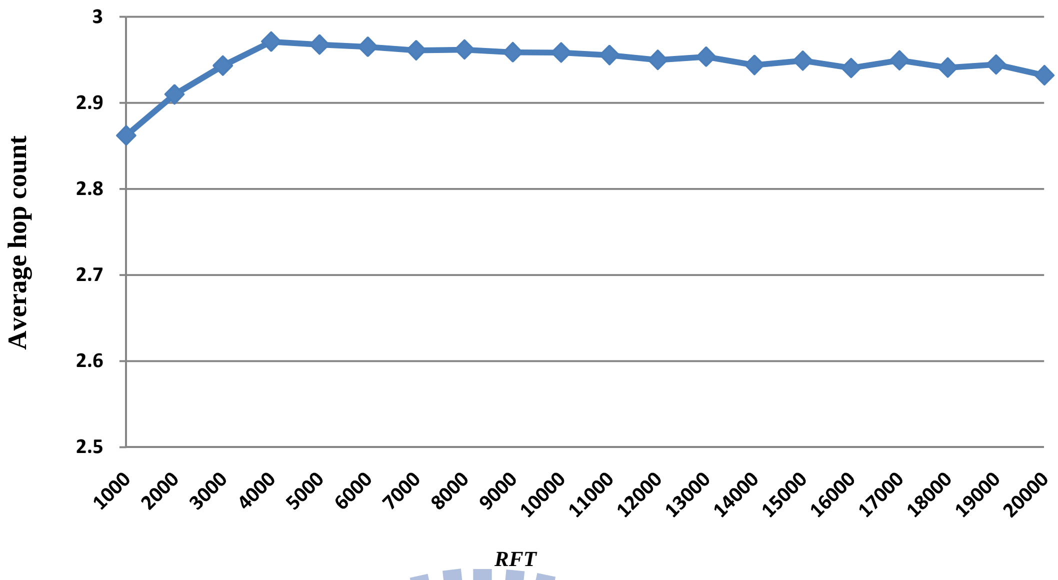


Figure 12. The average hop count of finding a target peer to publish an index under a different *RFT*.

Second, the number of search messages will also increase after applying the proposed KAD-mod method. However, the number of publish messages will not be affected. Note that total network messages include search messages and publishing messages. The percentage of extra traffic (T_e) for KAD-mod is defined as:

$$T_e = \frac{(Total\ network\ traffic\ of\ \ KAD\text{ - }mod) - (Total\ network\ traffic\ of\ \ KAD)}{Total\ network\ traffic\ of\ \ KAD}$$

In Figure 13, the percentage of extra traffic decreases with a higher *RFT*. However, *G* increases with a higher *RFT*, as shown in Figure 11. We found that 6000 is the optimal *RFT* in the proposed KAD-mod. Remind that the percentage of extra traffic for KAD-mod is small (8% for *RFT* = *6000*) number of search messages is much smaller than the number of publish messages.

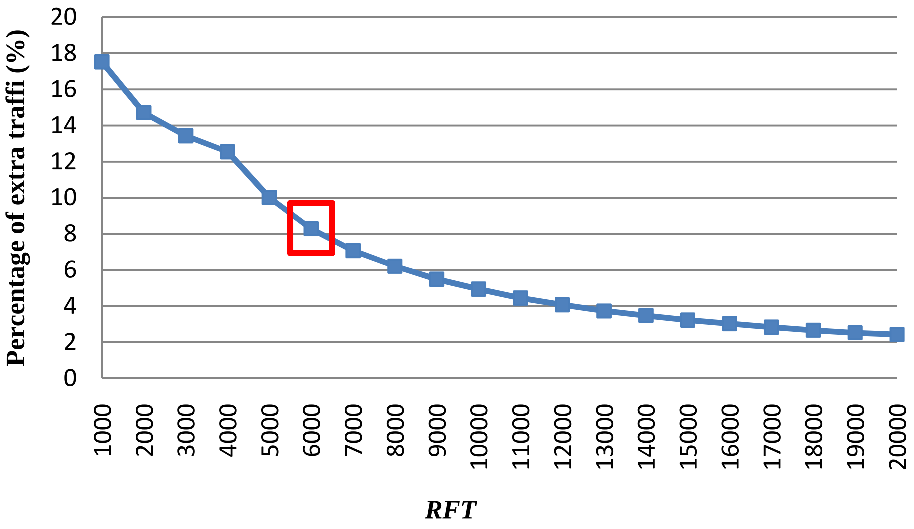


Figure 13. The percentage of extra traffic under a different *RTF*.

4.3 Comparison with existing load balancing schemes

KAD-mod can publish popular indexes more balanced than KAD-7 and KAD. The proposed KAD-mod can publish indexes to all of 256 zones when the number of the indexes in the original publish target peer exceeds *RFT*, while KAD-7 and KAD can only publish indexes to seven zones and one zone, respectively. Figure 14 shows the comparison of the Gini coefficient regarding the number of indexes in each zone among KAD-mod, KAD-7 and KAD. We found that KAD-mod is more balanced than KAD-7 and KAD. Figure 15(a) shows the percentage of extra traffic compared to KAD. KAD-mod has only 0.68% more extra network traffic than KAD-7. In Figure 15(b), we observed that KAD-mod's average publish hop count is only 0.5 hop more than KAD and KAD-7.

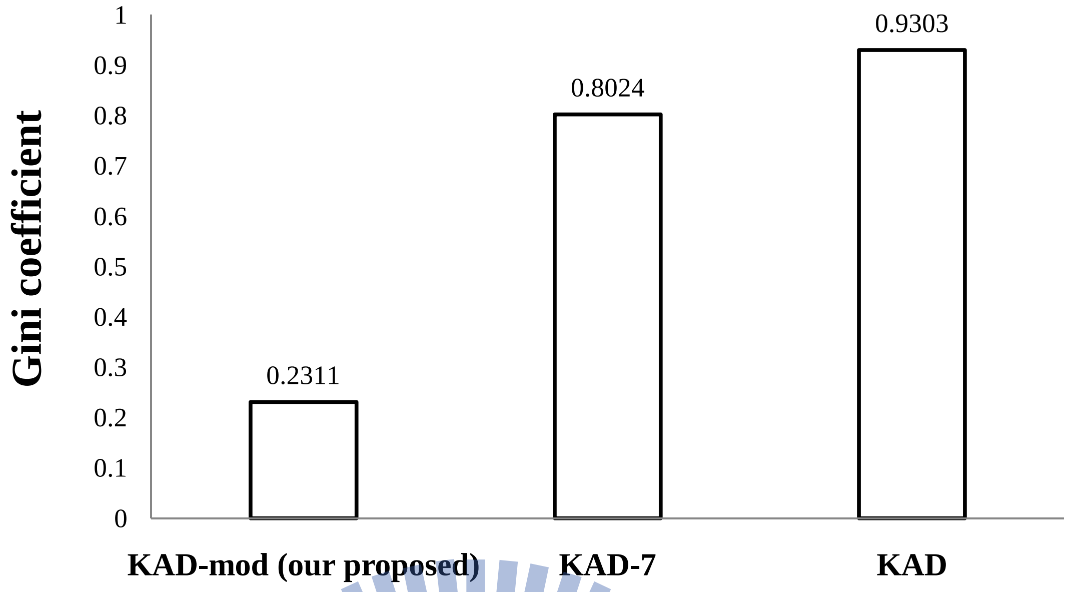
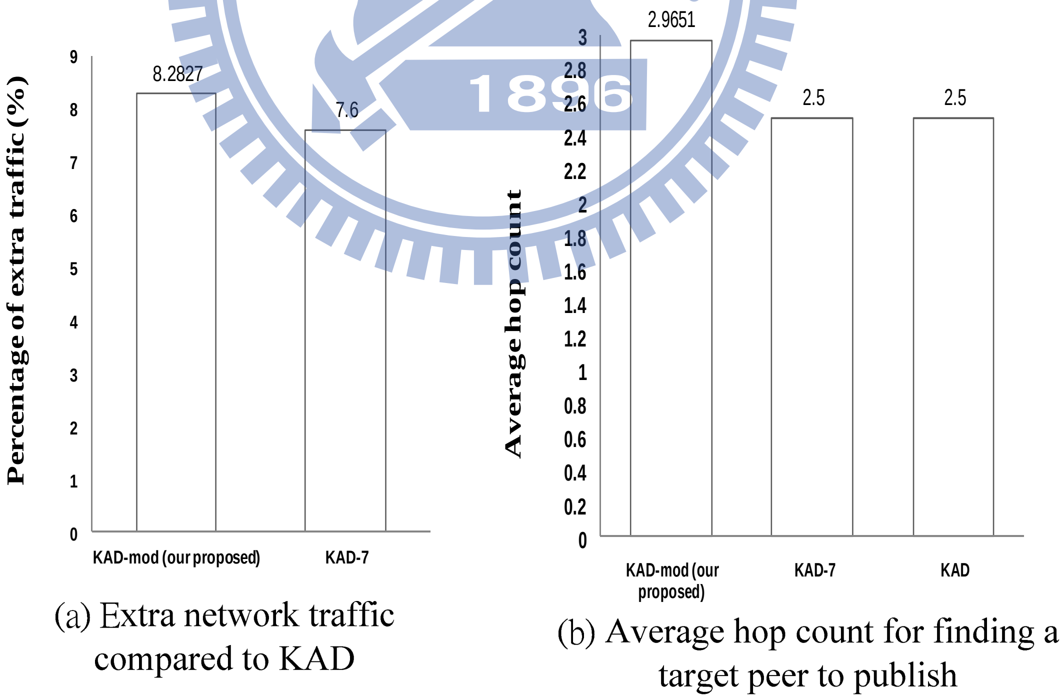


Figure 14. Comparison of publish load balancing among the three schemes in terms of the Gini coefficient.



(a) Extra network traffic compared to KAD

(b) Average hop count for finding a target peer to publish

Figure 15. Comparison of extra network traffic and average hop count for finding a target peer to publish.

Indexes in failed peers are called missing indexes. Objects referenced by missing indexes are not searchable. The search hit rate is calculated by 1- (dividing the number of missing indexes to the number of total indexes). According to [25], the percentage of failed peers in a day is about 27%. Figure 16 shows that the search hit rates of KAD-mod and KAD-7 are 98.24% and 97.71% when 27% of peers failed.

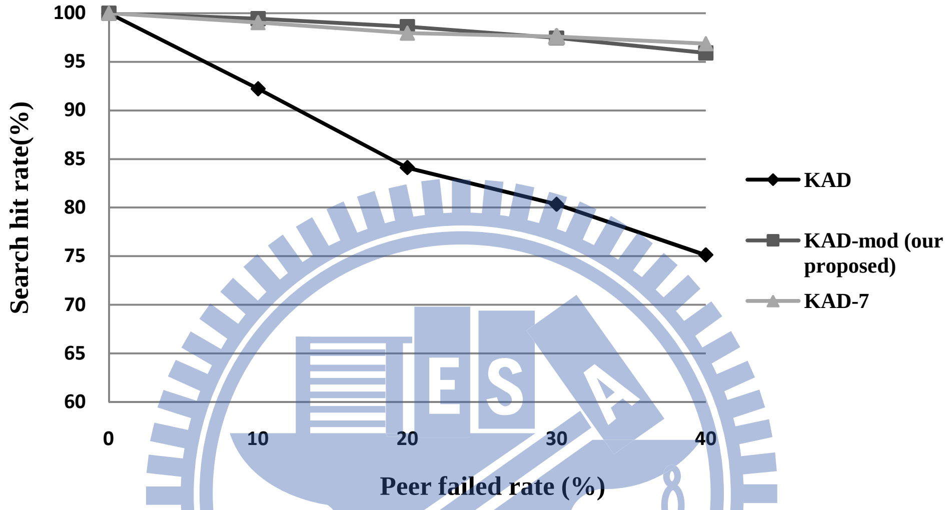


Figure 16. The search hit rate with respect to the peer failed rate for different approaches.

The publish load will affect the request load. We also evaluate the load balancing of requests. Since MHF [24] did not describe how to publish indexes, we only include MHF [24] for load balancing requests of comparison here. MHF [24] set the threshold of the request rate to 800 requests per second. Figure 17 shows *G* of the request load under the best, average, and worst cases of MHF. In the best case, all requests are from the same peer and the request rate is higher than the threshold of request rate all the time. The best case is almost impossible to happen because it does not meet the real P2P network characteristics. In the worst case, the request rates of all requests are lower than the threshold of request rate and this also does not meet the real P2P network characteristics. The average case can reflect the real P2P network characteristics.

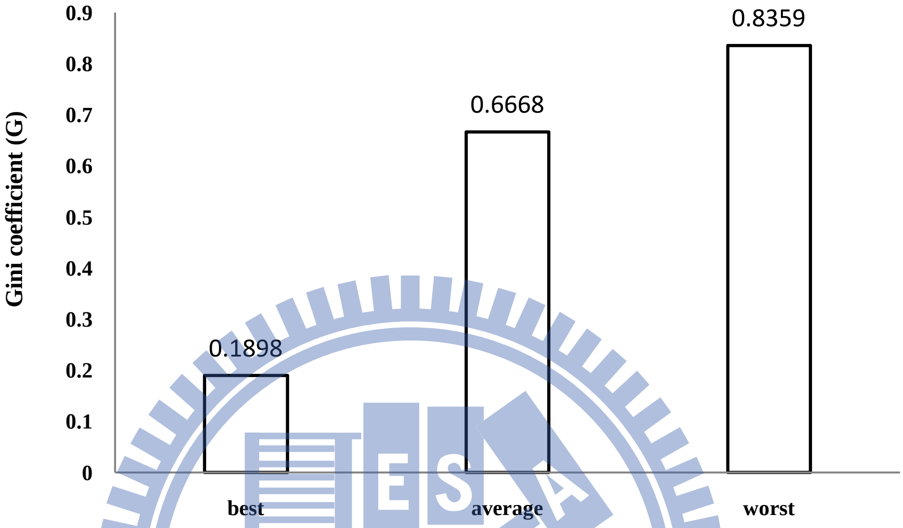


Figure 17. *G*'s of the request load for the best, average, and worst cases of MHF.

For the proposed KAD-mod, because indexes are evenly published, the request load will become even as well. Figure 18 shows the comparison of *G*'s regarding the request load among the four approaches. KAD-mod performs the best in terms of *G* of the request load, because in KAD-mod, the more popular indexes are handled by more peers.

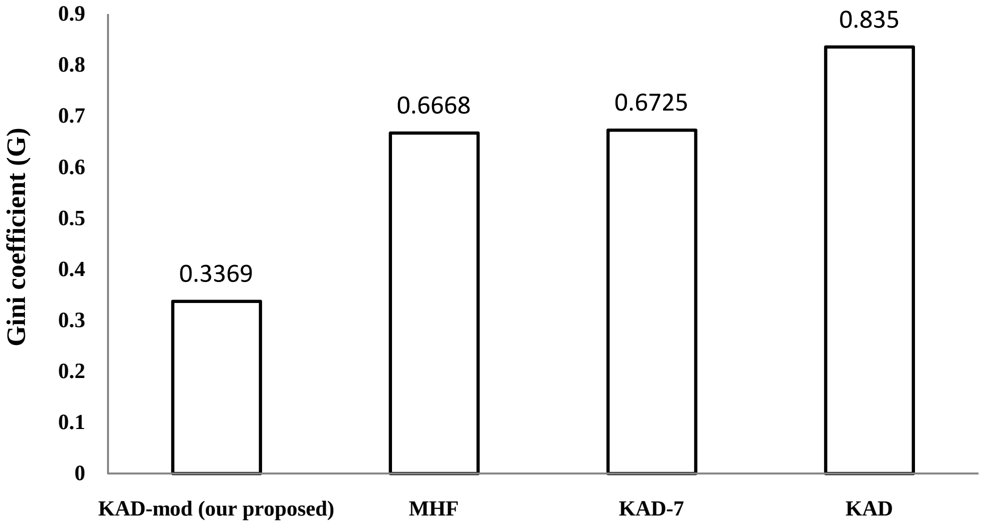


Figure 18. Comparison of *G*'s regarding the request load for four representative approaches.

Chapter 5 Conclusion

5.1 Concluding remarks

In this paper, we have presented an efficient modulo based method (KAD-mod) to balance the publish load and request load of KAD P2P networks. Our approach also improves the hit rate of keyword searching. The proposed KAD-mod is a simple and effective method without complex calculations. By redirecting overloaded indexes, indexes can be distributed more even, and not only the publish load but also the request load of each peer would be more balanced. Although the average hop count of finding a target to publish an index will increase and the total network traffic will slightly increase, these overhands are very small. Based on the simulation results, the G (G, $0 \leq G \leq 1$, 0: fully balanced) of publishing load for

KAD-mod is 0.23, KAD-7 is 0.80, and KAD is 0.93. As to *G* of request load, KAD-mod is 0.33, KAD-7 is 0.67, and KAD is 0.83. KAD-mod improves the search hit to 98% and only causes 8% extra traffic and KAD-mod‘s is only 0.5 hop more than KAD and KAD-7. Our method can not only improve the search resilience but also balance the publish and request load among peers in KAD P2P networks.

5.2 Future work

The proposed KAD-mod is simple and effective method to achieve publish load balancing, request load balancing, and search resilience. In the future, we will adapt our method to let it be applicable to other DHT based P2P networks. In addition, if the number of indexes become too large, how to flexibly adjust *RFT* to balance load in the KAD P2P network is deserved to further study.

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] "The Gnutella 0.4 protocol specification, 2000." [Online]. Available: http://dss.clip2.com/GnutellaProtocol04.pdf

[3] A. Oram et al., Peer-to-Peer: Harnessing the Power of Disruptive Technologies, 2001, O'Reilly.

[4] I.Clake, TW. Hong, O.Sanberg, and B.Wiley. "Protecting free expression online with freenet," *IEEE Trans. Internet Computing*, vol. 6, no. 1, pp.40-49, 2002.

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

[6] 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.*

[7] Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. ACM Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp. 161-172, August 2001.

[8] Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 161-172, August 2001.*

[9] 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.

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

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

[12] “aMula Project,” [Online]. Available: http://www.amule.org/

[13] B. T. Loo, J. M. Hellerstein, R. Huebsch, S. Shenker, and I. Stoica. “Enhancing P2P File-Sharing with an Internet-Scale Query Processor,” *Proceedings of the Thirtieth international Conference on Very large data bases*, vol. 30, pp. 432-443, 2004.

[14] Loo, b. T., Huebsch, r., Stoica,i., & Hellerstein, j. (2004). The Case for a Hybrid P2P Search Infrastructure. *In Proceedings of the 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, pp. 141-150, February 2004.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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

[20] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, “Load balancing in

dynamic structured p2p systems," in *Proceedings of the IEEE INFOCOM*, pp. 2253-2262, March 2004.

[21] M. Steiner, T. En-Najjary, and E. Biersack. "Long Term Study of Peer Behavior in the KAD DHT," in *Proceedings of the IEEE/ACM Transactions on Networking*, 2009.

[22] T. Pitoura , P. Triantafillou , T. Pitoura , P. Triantafillou. "Load Distribution Fairness in P2P Data Management Systems," in *Proceedings of the IEEE 23rd International Conference on Data Engineering*, pp. *396-405*, April 2007.

[23] T.T. Wu, K.C. Wang. "An Efficient Load Balancing Scheme for Resilient Search in KAD Peer to Peer Networks," in *Proceedings of the Ninth IEEE Malaysia International Conference on Communications*, pp. 759-764, March 2010.

[24] Y.Mu, C. Yu, T. Ma, C. Zhang, W. Zheng, X. Zhang. "Dynamic Load Balancing with Multiple Hash Functions in Structured P2P System," in *Proceeding of WiCom '09. 5th International Conference on Wireless Communications, Networking and Mobile Computing* , October 2009.

[25] 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.

[26] E.K.Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Trans. Communications Surveys & Tutorials, IEEE*, vol. 7, no. 2, pp. 72-93, 2005

[27] "Internet Study 2007," [Online]. Available: http://www.ipoque.com/

[28] Y. Zhu , Y. Hu. "Efficient, Proximity-Aware Load Balancing for DHT-Based P2P Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 4, pp. 349-361, 2005

[29] M. CASTRO, M. COSTA AND A. ROWSTRON,, "Peer-to-Peer overlays: *structured , unstructured, or both?*," Microsoft Research, Cambridge, CB3 0FB, UK, 2004.