



THESIS

7

054

60334612

This is to certify that the  
dissertation entitled

Overlay Topology Optimization and Security Studies in Peer-  
to-Peer Systems

presented by

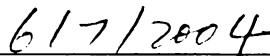
YUNHAO LIU

has been accepted towards fulfillment  
of the requirements for the

Ph.D. degree in Computer Science and Engineering



Major Professor's Signature



Date

*MSU is an Affirmative Action/Equal Opportunity Institution*

**LIBRARY**  
**Michigan State**  
**University**

**PLACE IN RETURN BOX** to remove this checkout from your record.  
**TO AVOID FINES** return on or before date due.  
**MAY BE RECALLED** with earlier due date if requested.

DATE DUE	DATE DUE	DATE DUE

# **Overlay Topology Optimization and Security Studies in Peer-to-Peer Systems**

**BY**

**Yunhao Liu**

**A DISSERTATION**

**Submitted to  
Michigan State University  
in partial fulfillment of the requirements  
for the degree of**

**DOCOTOR OF PHILOSOPHY**

**Department of Computer Science and Engineering**

**2004**

# Abstract

## Overlay Topology Optimization and Security Studies in Peer-to-Peer Systems

BY

*Yunhao Liu*

Current and future Internet and distributed systems rely on both centralized client-server model and decentralized peer-to-peer (P2P) model. P2P model is an emerging technology aiming to effectively utilize and manage increasingly large and globally distributed information and computing resources, complementing the available client-server services. In order to truly adopt the P2P model for deploying large-scale Internet applications, and timely merge this model as an indispensable component in the main stream of distributed computing technology, we must address several major technical challenges including the efficiency of overlay networks, cost-effective P2P information search, and privacy and security protection of peers. This dissertation focuses on addressing two critical issues. The first issue is topology mismatch problem between P2P overlay networks and the underlying physical networks in unstructured P2P systems. Addressing topology mismatch problem can fundamentally improve overall search performance of P2P systems. We demonstrate the seriousness of the topology mismatch problem, and define an optimal overlay problem that is proved to be a NP-hard problem. We then develop several effective schemes and algorithms to alleviate the topology mismatch problem. Our proposed algorithms are completely distributed, scalable and effective. Simulation studies show that the total traffic and response time of the queries can be significantly reduced by these schemes without shrinking the search scope. The second issue is overlay

distributed denial-of-service (DDoS) attack in P2P systems. Most previous security techniques protect networks from network-layer DDoS attacks, but cannot be applied to overlay DDoS attacks. We propose a distributed and scalable method, DD-POLICE, to detect malicious nodes in order to defend P2P systems from overlay flooding-based DDoS attacks. We show the effectiveness of DD-POLICE by simulation studies and implementation on Gnutella 0.6 protocols. We believe that widely employing these proposed approaches will make P2P systems more scalable and robust.

**to my grandmother**

## ACKNOWLEDGMENT

First and foremost, I would like to deeply thank my advisor Dr. Li Xiao for her continual guidance and helping me throughout. I also want to thank Dr. Lionel Ni, Dr. Abdol-Hossein Esfahanian, and Dr. Xiaodong Zhang for their support and great advice during this work. I am thankful to Dr. Matt W. Mutka and Dr. Z. J. Wang for their suggestions and comments.

I would like to thank my grandmother and my parents for their unconditional love and encouragement. They always trust me more than I did. I am also grateful to the members of the ELANS lab especially Abhishek Patil, Xiaomei Liu, Chen Wang, and Zhenyun Zhuang for exchanging ideas.

## TABLE OF CONTENTS

1	Introduction .....	1
1.1	<i>Research Background</i> .....	1
1.2	<i>Problem Statement and Research Objectives</i> .....	4
1.2.1	Search efficiency.....	4
1.2.2	Overlay distributed denial-of-service.....	5
1.3	<i>Contributions</i> .....	6
1.4	<i>Thesis Organization</i> .....	7
2	Related Work.....	9
2.1	<i>Improving Search Efficiency of P2Ps</i> .....	9
2.2	<i>Defending against Distributed Denial of Service</i> .....	13
3	Simulation Methodology of Dynamic P2P Environments .....	16
3.1	<i>Performance Metrics</i> .....	16
3.2	<i>Simulation Methodology</i> .....	18
3.2.1	Flooding search simulation .....	19
3.2.2	A dynamic P2P environment .....	20
4	Overlay Optimization in P2P Systems .....	22
4.1	<i>Optimal Overlay Problem</i> .....	22
4.1.1	Message Duplications in Overlay Connections .....	23
4.1.2	Message Duplications in Physical Links and Topology Mismatch Problem .....	25
4.1.3	Modeling P2P Networks .....	27
4.1.4	Amount of Message Duplications in Overlay Connections.....	31
4.1.5	Optimal Overlay Problem .....	33
4.2	<i>Adaptive Overlay Topology Optimization (AOTO)</i> .....	35
4.2.1	Selective Flooding.....	36
4.2.2	Active Topology.....	37
4.2.3	Effectiveness of Select Flooding Procedure .....	40
4.2.4	Effectiveness of Active Topology Procedure .....	44
4.2.5	Convergent Speed of AOTO.....	48
4.2.6	AOTO in Dynamic Environments .....	49
4.3	<i>Location-aware Topology Matching (LTM)</i> .....	51
4.3.1	TTL2-detector flooding.....	51
4.3.2	Low productive connection cutting.....	52
4.3.3	Source peer probing .....	54

4.3.4	Traffic Overhead of LTM .....	55
4.3.5	Effectiveness of LTM in Static Environment .....	56
4.3.6	LTM in Dynamic Environment.....	59
4.3.7	Combining LTM and Query Index Caching .....	65
4.4	<i>Scalable Bipartite Overlay (SBO)</i> .....	67
4.4.1	Design of SBO .....	67
4.4.2	Further Improvements .....	74
4.4.3	Traffic Overhead of SBO Optimizations .....	76
4.4.4	Property analysis of SBO operations .....	76
4.4.5	Effectiveness of SBO in Static Environments .....	78
4.4.6	Frequency of SBO Optimizations .....	80
4.4.7	SBO with SDC and Index Caching.....	83
4.5	<i>Two Hop Away Neighbor Comparison and Selection (THANCS)</i> .....	85
4.5.1	Piggyback Neighbor Distance on Queries .....	85
4.5.2	Neighbor Comparison and Selection .....	89
4.5.3	Effectiveness Analysis of THANCS .....	91
4.5.4	Effectiveness of THANCS in Static Environments .....	92
4.5.5	THANCS in Dynamic Environments .....	94
4.5.6	Effectiveness with other advanced search strategies .....	101
4.6	<i>Discussion</i> .....	103
5	<i>Defending P2Ps from Overlay Flooding-based DDoS</i> .....	106
5.1	<i>Definition of a “good” Peer and a “bad” Peer</i> .....	108
5.2	<i>An Implementation of an Overlay DDoS Agent</i> .....	111
5.2.1	Query Trace Collection .....	111
5.2.2	An Implementation of an Overlay DDoS Agent.....	111
5.3	<i>Design of DD-POLICE</i> .....	113
5.3.1	Neighbor List Exchanging .....	115
5.3.2	Neighbor Query Traffic Monitoring .....	117
5.3.3	Bad peer recognition .....	117
5.3.4	An example of DD-POLICE .....	120
5.3.5	DD-POLICE-r .....	122
5.4	<i>Performance Metrics</i> .....	123
5.5	<i>Simulation Results</i> .....	124
5.5.1	Consequences of overlay DDoS attack in P2Ps.....	124
5.5.2	Effectiveness of DD-POLICE.....	127
5.5.3	DD-POLICE in Dynamic P2P Environments .....	131
5.6	<i>Implementation of DD-POLICE</i> .....	132
5.7	<i>Discussion</i> .....	134
6	<i>Conclusion and Future Work</i> .....	136

6.1	<i>Summary</i> .....	136
6.2	<i>Future Work</i> .....	139

## LIST OF TABLES

Table 1: TTL2-detector message body .....	52
Table 2: Piggy Message body .....	85
Table 3: Neighbor Traffic message body .....	118

## LIST OF FIGURES

Figure 1.1 A centralized P2P network.....	2
Figure 1.2 A decentralized unstructured P2P network.....	3
Figure 2.1 A Query result caching scheme .....	11
Figure 2.2 before optimization, queries can visit all of the peer, while after optimization, all queries can only visit a small group of live peers .....	13
Figure 4.1 Bootstrapping a new peer in Gnutella.....	23
Figure 4.2 Examples of P2P overlay topologies .....	24
Figure 4.3 Examples of physical topologies .....	25
Figure 4.4 Percentage of query responses along mismatched paths .....	26
Figure 4.5 A super peer behaviour observation experiment.....	29
Figure 4.6 Query drop rate vs. query density .....	31
Figure 4.7 Selective Flooding .....	37
Figure 4.8 Active Topology .....	39
Figure 4.9 ORs and OR distribution on a 1000-node logical topology.....	41
Figure 4.10 Average ORs on 10 physical topologies.....	42
Figure 4.11 Average ORs on 20 logical topologies with different number of nodes.	42
Figure 4.12 OR for different average neighbor numbers .....	43
Figure 4.13 Optimization for 3 Ways.....	45
Figure 4.14 Topology degree properties changes.....	45
Figure 4.15 Cost optimization .....	46
Figure 4.16 Threshold factors.....	46
Figure 4.17 Traffic reduction vs. optimization step in AOTO .....	47
Figure 4.18 Response time reduction vs. optimization step in AOTO.....	48
Figure 4.19 Traffic reduction of AOTO in dynamic P2P environment .....	49

Figure 4.20 Average response time reduction of AOTO in dynamic P2P environment .....	50
Figure 4.21 Peer P receives $d(i, S, v)$ multiple times in LTM.....	53
Figure 4.22 Source peer probing .....	54
Figure 4.23 Traffic cost vs. search scope .....	57
Figure 4.24 Traffic reduction vs. optimization step .....	57
Figure 4.25 Average neighbor distance vs. optimization step.....	58
Figure 4.26 Average response time vs. optimization step.....	59
Figure 4.27 Average search latency vs. optimization step .....	60
Figure 4.28 Effectiveness of will-cut list .....	61
Figure 4.29 Effectiveness of cut list.....	62
Figure 4.30 Total traffic vs. LTM frequency .....	63
Figure 4.31 Response time vs. LTM frequency .....	63
Figure 4.32 Optimal LTM frequency vs. average peer lifetime.....	64
Figure 4.33 Optimal LTM frequency vs. average query frequency .....	65
Figure 4.34 Traffic cost of four schemes .....	66
Figure 4.36 Bootstrapping a new peer.....	68
Figure 4.37 A red peer P has topology of $(P+N(P) + N^2(P))$ , and computes the MST .....	69
Figure 4.38 A red peer computes the efficient forwarding paths.....	70
Figure 4.39 An example of neighbor replacement .....	73
Figure 4.40 Proof of the property of SBO operations .....	77
Figure 4.41 Traffic reduction vs. optimization steps.....	79
Figure 4.42 Response time vs. optimization steps .....	79
Figure 4.43 Traffic cost reduction of SBO in dynamic P2P environments.....	81
Figure 4.44 Response time reduction in dynamic P2P environments.....	81

Figure 4.45 Optimal SBO optimization time interval .....	82
Figure 4.46 Traffic cost reduction of SBO in dynamic P2P environments.....	84
Figure 4.47 Response time reduction of SBO in dynamic P2P environments.....	84
Figure 4.48 Forwarding piggy messages.....	86
Figure 4.49 New neighbor triggered policy .....	87
Figure 4.50 Probing two hop away neighbors.....	89
Figure 4.51 THANCS can effectively optimize a mismatched overlay topology to a better mapping overlay. (a) With inefficient mapping, a broadcast message issued by node A travels physical link D-E six times. (b) After THANCS optimization, with a better mapping, the same message traverses D-E link only once. ....	92
Figure 4.52 The percent of query responses along mismatched paths of three schemes .....	93
Figure 4.53 Average traffic cost per query.....	95
Figure 4.54 Average query response time.....	95
Figure 4.55 Performance of THANCS with NNT or PPB policies on reducing mismatch degree.....	96
Figure 4.56 Traffic cost overhead of THANCS with NNT or PPB policies.....	97
Figure 4.57 Traffic cost reduction of THANCS in different size overlays ranging from 2,000 to 8,000 nodes.....	99
Figure 4.58 Average response time reduction of THANCS in different size overlays ranging from 2,000 to 8,000 nodes.....	99
Figure 4.59 Traffic cost reduction of THANCS in 5,000-node overlays with different average number of neighbors ranging from 4 to 6 .....	100
Figure 4.60 Average response time reduction of THANCS in 5,000-node overlays with different average number of neighbors ranging from 4 to 6 .....	101
Figure 4.61 Traffic cost and average response time reduction on Random Walk and Index Cache schemes when employing THANCS.....	102
Figure 4.62 Traffic cost reduction of AOTO, LTM, SBO and THANCS .....	103
Figure 4.63 Response time reduction of AOTO, LTM, SBO and THANCS.....	104

Figure 5.1 Although peer q sends out 5,000 queries per minute, it is a “good” peer	107
Figure 5.2 Query traffic analysis .....	110
Figure 5.3 Experiment of implementing a DDoS agent peer. Peer A is a “bad” peer, which may send out a large volume of queries continuously; Peer B is a normal “good” peer, which forwards out as many of the received queries as it is capable of; Peer C is our query traffic observer, which counts all the queries forwarded by B, and does not issue or forward any query .....	112
Figure 5.4 Queries sent out vs. processed .....	114
Figure 5.5 Neighbor List Exchanging .....	115
Figure 5.6 An example of a Buddy Group $BG1-j=\{A,B,C,D\}$ .....	116
Figure 5.7 An example of DD-POLICE.....	120
Figure 5.8 Average traffic cost.....	125
Figure 5.10 Success rate.....	127
Figure 5.11 Effectiveness of DD-POLICE in Dynamic P2P environments.....	129
Figure 5.13 Damage recovery time vs. cut threshold.....	131
Figure 5.14 Prototype of a DD-POLICE enabled client .....	132
Figure 5.15 A simple experiment of DD-POLICE.....	133
Figure 5.16 Experimental Results of DD-POLICE implementation. Both peer Q and peer O may successfully disconnect with the malicious peer A within 15 seconds from when peer A starts flooding a large volume of queries to them	134

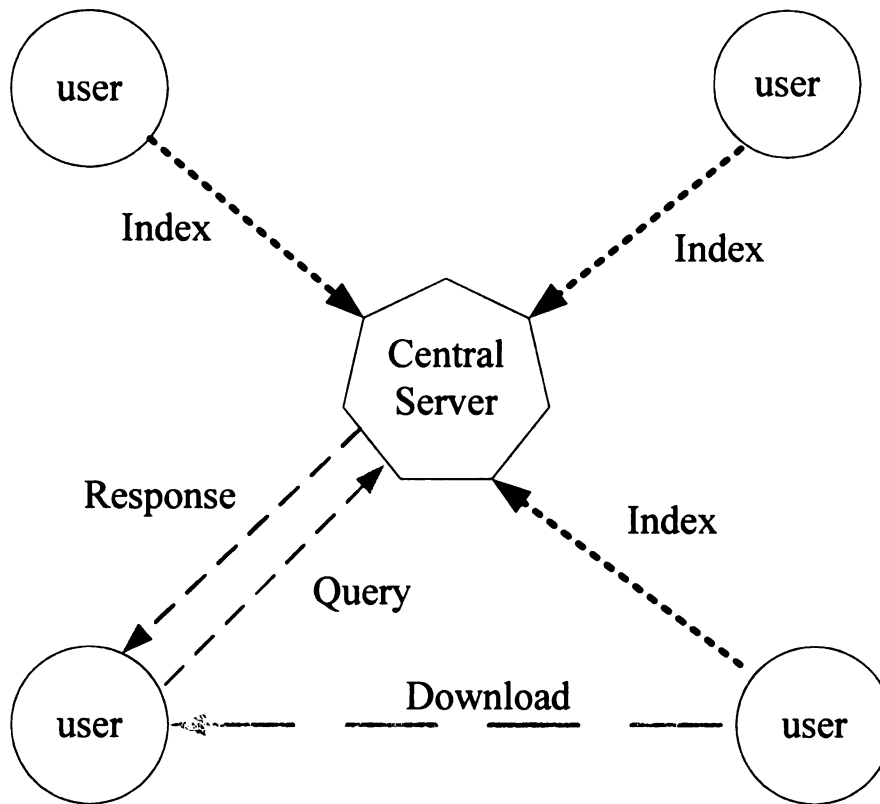
# 1 Introduction

Since the emergence of peer-to-peer (P2P) file sharing applications, such as Napster[12], Gnutella[8], and KaZaA[10], millions of users have started using their home computers for more than browsing the web and exchanging Emails. Each peer acts as both a client who requests information and services and a server who produces and/or provides information and services.

## *1.1 Research Background*

There are mainly three different architectures for P2P systems: centralized, decentralized structured, and decentralized unstructured [55]. In the centralized model, such as Napster [12], as shown in Figure 1.1, a central index server is used to maintain a directory of shared files stored on peers so that a peer can search for the whereabouts of a desired content from the index server, and download the content directly from the peer who has the content. However, this architecture creates a single point of failure and its centralized nature of the service also makes systems vulnerable to denial of service attacks. Decentralized P2P systems have the advantages of eliminating reliance on central servers and

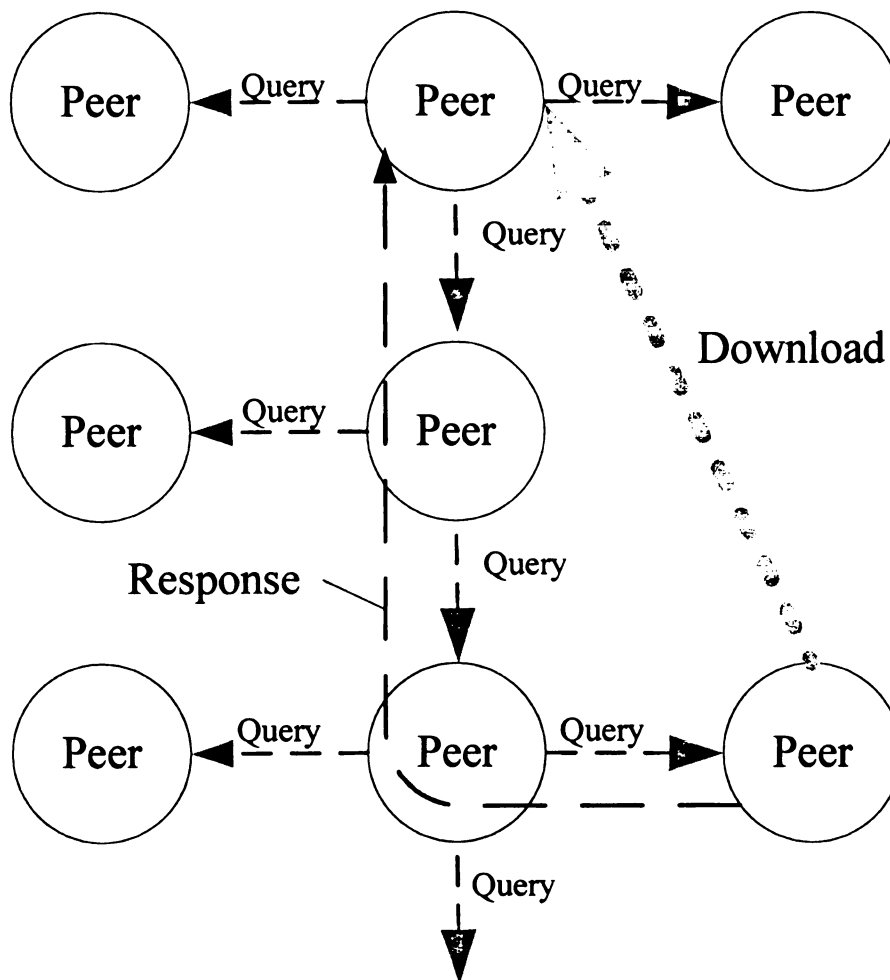
providing greater freedom for participating peers to exchange information and services directly between each other.



**Figure 1.1 A centralized P2P network**

In decentralized structured models, such as Chord [84], Pastry [71], Tapestry [96], and CAN [67], the shared data placement and topology characteristics of the network are tightly controlled based on distributed hash functions. Although these designs are expected to dramatically improve the search performance, none of them has been practically used due to their high maintenance traffic in delivering messages and updating the mapping. Furthermore, it is hard for structured P2P systems to efficiently support partially matched queries.

My research focuses on decentralized unstructured P2P systems, such as Gnutella [8] and KaZaA [10]. File placement is random in these systems, which has no correlation with the network topology [94]. Unstructured P2P systems are most commonly used in today's Internet. The most popular search mechanism in use is to blindly "flood" a query to the network among peers (such as in Gnutella) or among super peers (such as in KaZaA), as shown in Figure 1.2.



**Figure 1.2 A decentralized unstructured P2P network**

A query is broadcasted and rebroadcast until a certain criterion is satisfied. If a peer receiving the query can provide the requested object, a response message will be sent back to the source peer along the inverse of the query path. This mechanism ensures that the queries are “flooded” to as many peers as possible within a short period of time in a P2P overlay network. A query message will also be dropped if the query message has visited the peer before.

## ***1.2 Problem Statement and Research Objectives***

Unstructured peer-to-peer models are simple to implement and widely used in real systems. However, there are efficiency and security issues to be addressed.

### **1.2.1 Search efficiency**

Studies in [78] and [73] have shown that P2P traffic contributes the largest portion of the Internet traffic based on their measurements on some popular P2P systems, such as FastTrack (including KaZaA and Grokster) [7], Gnutella, and DirectConnect. Measurements in [69] have shown that even given that 95% of any two nodes are less than 7 hops away and the message time-to-live (TTL=7) is preponderantly used, the flooding-based routing algorithm generates 330 TB/month in a Gnutella network with only 50,000 nodes. A large portion of the heavy P2P traffic caused by inefficient overlay topology and the blind flooding is unnecessary, which makes the unstructured P2P systems being far from scalable [70]. There are three reasons for this problem. First, the mechanism of a peer randomly choosing logical neighbors without any knowledge about the underlying physical topology causes topology mismatching between the P2P logical overlay network and physical underlying network. Because of the mismatch problem, the same message may traverse the same physical link multiple times, incurring a large amount of unnecessary

traffic. Second, a query may be flooded to multiple paths that are merged to the same peer. In this case, only the traffic along one of the paths is necessary. Finally, two neighboring peers may forward the same query message to each other before they receive the query message from each other. Thus, the same query message may traverse the same logical link twice.

Aiming at alleviating the mismatch problem, reducing the unnecessary traffic, and addressing the limits of existing solutions, we propose four topology optimization based approaches, including adaptive overlay topology optimization (AOTO), location-aware topology matching (LTM) scheme, Scalable Bipartite Overlay (SBO), and Two Hop Away Neighbor Comparison and Selection (THANCS), to make the decentralized unstructured P2P models more scalable and efficient.

### **1.2.2 Overlay distributed denial-of-service**

The simplicity of the flooding based search mechanism also makes unstructured P2Ps vulnerable to overlay distributed denial-of-service (DDoS) attacks. In the past years, DDoS attacks have already become a major threat to the stability of the Internet [22]. The basic goal of denial of service (DoS) is to overwhelm the processing or link capacity of the target by saturating it with bogus packets. Flooding based overlay DDoS attacks are DoS attacks performed from multiple compromised peers (agents), who start generating as many bogus queries as they can toward the victims. One character of P2P overlay DDoS is that the attack target is not a single site or a user in the Internet, but the whole P2P systems.

Most previous research on DDoS focused on Network layer attacks, including direct attacks such as TCP-SYN, ICMP flooding and UDP flooding [22, 77], and reflector at-

tacks such as Smurf attacks [2]. Many attack tools such as Trinoo, Tribe Flood Network, TFN2K, Stacheldraht, Shaft, mstream [4-6] have been deployed. Existing approaches mainly fall into three categories: prevention, traceback and identification, and detection and filtering. Since a P2P system can have millions of insecure users online simultaneously, and the IP address of a query source peer is not included in the query or query hit messages, network layer defense approaches often find it difficult, if not outright impossible, to effectively protect against overlay DDoS attacks. It is therefore a worthwhile endeavor to design an overlay level defending mechanism inside P2P applications. In this research, a detection-based approach, DD-POLICE (**Defending P2Ps from Overlay Distributed-Denial-of-Service**), is proposed, to protect P2P systems against overlay DDoS.

### ***1.3 Contributions***

The main contributions of this research are as follows.

First, we model the unstructured P2P systems based on our observations and implementations of Gnutella peers. We then study the relationship between the property of the overlay and the corresponding message duplications incurred by queries in a given overlay, and prove that even with global knowledge, computing an optimal overlay is an NP-hard problem. We demonstrate that a large portion of the traffic of today's widely used peer-to-peer systems is unnecessary.

Second, we proposed several (AOTO, LTM, SBO and THANCS) overlay topology matching algorithms, and simulated them in both static and dynamic environments. They are all completely distributed and scalable in that they do not need any global knowledge, and each peer conducts the algorithm independently. Compared with traffic cost savings, the overhead incurred by these algorithms is trivial. The other strength of these algo-

algorithms is that they are complementary with other cache based and forwarding based approaches and can be deployed together.

Third, we show the difficulties of defending overlay DDoS in P2Ps and the reasons why existing network layer defense approaches are less effective on overlay DDoS attacks. To investigate the characteristics of flooding based overlay DDoS, we modified LimeWire Gnutella servant with support of Gnutella protocol v0.6 [9] to collect real Gnutella query traffic trace. We also develop a prototype of an overlay DDoS agent who may continuously sends out a large amount of queries into the P2P network. We then propose DD-POLICE, which requires peers to cooperate locally with their neighboring peers and identify malicious/ compromised DDoS peers effectively. By comprehensive simulations, we show the serious impact of overlay DDoS attacks on P2P systems, and the effectiveness of DD-POLICE in dynamic P2P environments. We then implement a prototype of a DD-POLICE enabled client, and show that DD-POLICE is easy to implement and effective on defending against overlay DDoS in P2P systems.

We believe this research will make the unstructured peer-to-peer systems more scalable, efficient, and secured.

## ***1.4 Thesis Organization***

The rest of this dissertation is organized as follows. Chapter 2 reviews existing approaches on improving search performance and defending against DDoS. The simulation methodology of the work is presented in Chapter 3. Overlay optimization approaches, including Adaptive Overlay Topology Optimization (AOTO), Location-aware Topology Matching (LTM), Scalable Bipartite Overlay (SBO), and Two Hop Away Neighbor Comparison and Selection (THANCS), are described in detail in Chapter 4. Security is-

sue studies and DD-POLICE approach for defending against overlay flooding based DDoS are presented in Chapter 5. The conclusion and future work of this research is in Chapter 6.

## 2 Related Work

In the past years, peer-to-peer systems have been under intensive studies[17, 19, 22, 24, 26-32, 35, 36, 38, 39, 42-44, 47-57, 63, 67-72, 78, 79, 83-85, 87-90, 92-94, 96, 97]. In this chapter, we will discuss related previous researches on search efficiency and security issues.

### *2.1 Improving Search Efficiency of P2Ps*

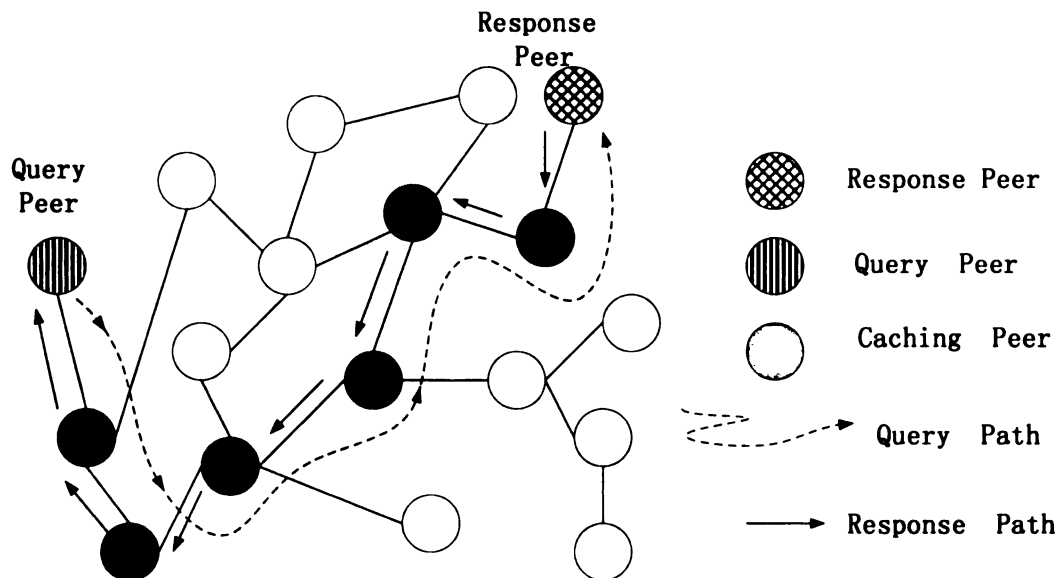
Many efforts have been made to avoid the large volume of unnecessary traffic incurred by the flooding-based search in decentralized unstructured P2P systems. In general, three types of approaches have been proposed to improve search efficiency in unstructured P2P systems: forwarding-based, cache-based, and overlay optimization. The three different kinds of approaches can be used together to complement each other.

In forwarding-based approaches, instead of relaying the query messages to all its logical neighbors except the incoming peer, a peer selects a subset of its neighbors to relay the query. In Directed BFS proposed in [94], each peer maintains statistic information based on some metrics, for example, the number of results received from neighbors from previous queries or the latency of the connection with that neighbor. A peer selects a sub-

set of the neighbors, such as the neighbors that have returned the largest number of results from previous queries, or the neighbors that have low latency, to send its query. A  $k$ -walker query algorithm is proposed in [55], in which a query is sent to  $k$  different walkers (relay neighbors) from the source peer. For a peer in each walker, it just randomly selects one neighbor to relay the query. For each walker, the query processing is done sequentially. A hybrid periodical flooding (HPF) approach proposed in [97] improves the search efficiency by selecting forwarding neighbors based on multiple metrics and addressing the partial coverage problem to balance the search cost and response time.

The second approach is cache-based including data index caching and content caching. Centralized P2P systems provide centralized index servers to keep indices of shared files of all peers. KaZaA utilizes cooperative superpeers, each of which is an index server of a subset of peers. Some systems distribute the function of keeping indices to all peers [57]. In Local Indices policy [94], each peer maintains an index of files available in the nodes within given hops of itself. When a peer receives a query, it can process the query on behalf of all nodes within the given hops of itself. Having observed the locality of queries, the authors in [56, 82] further proposed that each peer caches query strings and results that flow through it, which is shown in Figure 2.1. Three different strategies to replicate data (file content or query responses) on multiple peers have been evaluated in [28]. The three strategies are different on the ratio of allocations according to the ratio of query rates. Transparent query caching [65] is proposed to cache query hits at a gateway of an organization based on an observation of query locality in peers within the gateway. Caching file contents has also been studied. For example, an ideal cache (infinite capacity and no expiration) simulator [73] is built for KaZaA P2P traffic to cache file contents, which

has shown that caching would have a large effect on a wide-scale P2P system on reducing traffic volume and bandwidth demands.



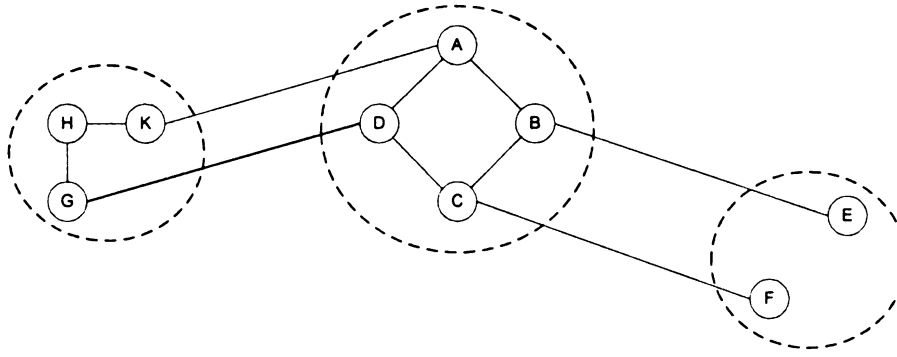
**Figure 2.1 A Query result caching scheme**

The third approach is based on overlay topology optimization that is closely related to what we are presenting in this dissertation. End system multicast, Narada, is proposed in [25], which first constructs a rich connected graph on which to further construct shortest path spanning trees. Each tree rooted at the corresponding source using well-known routing algorithms. This approach introduces large overhead of forming the graph and trees in a large scope, and does not consider the dynamic joining and leaving characteristics of peers. The overhead of Narada is proportional to the multicast group size. This approach is infeasible to large-scale P2P systems. Researchers have also considered to cluster close peers based on their IP addresses (e.g., [47, 64]). We believe there are two limitations for this approach. First, the mapping accuracy is not guaranteed by this ap-

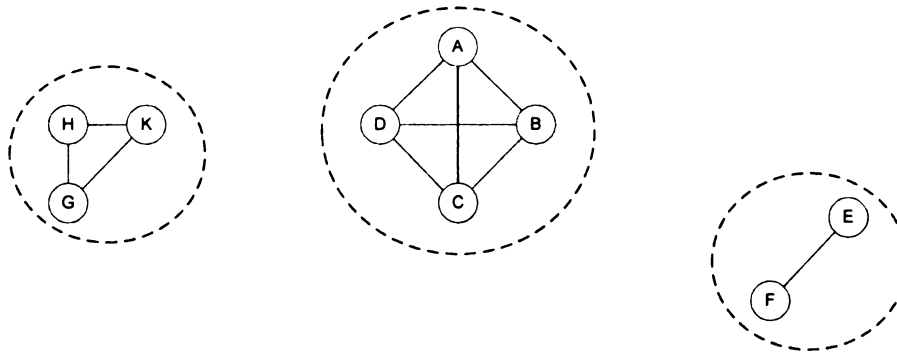
proach. Second, this approach may affect the searching scope in P2P networks. Recently, researchers in [90] have proposed to measure the latency between each peer to multiple stable Internet servers called “landmarks”. The measured latency is used to determine the distance between peers. This measurement is conducted in a global P2P domain and needs the support of additional landmarks. Similarly, this approach also affects the search scope in P2P systems.

Using an example shown in Figure 2.2, we explain why these existing proximity based approaches will shrink query search scopes. In Figure 2.2, peers A, B, C, D locate in the same AS, peers E, F and H, G, K belong to other ASs, respectively. It is safe to assume that the physical distance between A and B or E and F are much smaller than that of K and A or C and F, as illustrated in Figure 2.2. Using above discussed approaches, when peers successfully obtain or estimate the distance between each pair of them, and optimization policy for each node is to connect the closest peers while retaining the original number of logical neighbors, a connected graph may be broken into three components. As a result, before optimization, queries can visit all of the peer, while after optimization, all queries can only visit a small group of live peers in the system, and the search scope of queries is significantly reduced.

Gia [24] introduced a topology adaptation algorithm to ensure that high capacity nodes are indeed the ones with high degree and low capacity nodes are within short reach of high capacity nodes. It addresses a different matching problem in overlay networks, but does not address the topology mismatch problem between the overlay and physical networks.



Before Optimization



After Optimization

**Figure 2.2 before optimization, queries can visit all of the peer, while after optimization, all queries can only visit a small group of live peers**

## 2.2 *Defending against Distributed Denial of Service*

Many general efforts have been made to defend against DDoS [22, 46, 61, 62]. They are roughly divided into three categories: prevention, traceback and identification, and detection and filtering.

The first type is prevention, in which the defense tools monitor or scan the network to intercept the DDoS before the attacks start. They avoid the malicious peers' illegal access to normal machines [15], and install security patches and virus scanners [3, 45, 60]. One basic goal of these approaches is to prevent DDoS attackers from recruiting a large num-

ber of agents. Most of these methods are based on knowledge of existing DDoS attacks to recognize attackers' behaviors. Although it is of great importance to improve Internet security, it is hard to believe that preventive approaches could successfully avoid DDoS attackers recruiting hundreds of agents in a P2P network with millions of peers online at any given time. We will show that only tens of DDoS agents in a 20,000-peer system cause serious damage.

The second type is the TraceBack and Identification approach [74, 75, 80, 81, 91], which is usually employed after experiencing attacks. Most of them are based on IP traceback. They try to track the attackers and identify them via the routers' records or by sending special traceback packets. However, these approaches are not effective for P2P overlay DDoS attacks because the query messages and query hit messages do not include the IP addresses of query source peers. In P2P systems, the anonymity requirement makes it hard to know who originally issued the queries.

The third type is based on Detection and Filtering. Our proposed DD-POLICE is of this type. They detect the occurrence of DDoS attacks and respond to it. For example, although IP spoofing is not a necessary component for DDoS attacks, it helps the attackers hide [59]. Ingress/egress filtering [34] prevents packets with spoofed source IP addresses from entering or leaving the network. Theoretically, ubiquitous ingress packet filtering (UIPF) [22] can stop all address-spoofed direct attack packets as well as the attack packets sent to reflectors. But it is hard for them to be employed in P2P systems with such a large scale. Route-Based defense is similar to UIPF, and employs some distributed detection systems and filters attack traffic at some key nodes. Pushback mechanism [40] is a useful means of detecting the attack packets' flow and dropping them. In a P2P system, it

is hard for a peer to trace the source of a query because the values of TTL and hops could be easily modified by DDoS compromised peers. Thus, it is extremely hard to separate good traffic and bad traffic. A source-end defense system, such as D-WARD [58] or Reverse Firewall, attempts to observe incoming and outgoing flows and connections over time, aiming at separating the normal links from the attack links to provide good service to normal clients. However, in a P2P system, since normal traffic and attack traffic may come from the same logical link because of the flooding search, the source-end system is not effective in P2P systems.

To our knowledge, the most related work to this research to date is discussed in [30], where application-layer load balancing techniques are proposed to give clients a fair share of available resources, so as to alleviate damage of application-layer DoS attacks. It is basically a survival approach: it does not require servers to distinguish attack queries from normal queries, but maintain a fair load distribution in the P2P system. However, this approach could be less effective when the number of DDoS agents is getting large.

### 3 Simulation Methodology of Dynamic P2P Environments

We evaluate our proposed methods by comprehensive simulations and implementations. In this chapter, we present our performance metrics and simulation methodology.

#### 3.1 *Performance Metrics*

A well-designed search mechanism should seek to optimize both efficiency and Quality of Service (QoS). Efficiency focuses on better utilizing resources, such as bandwidth and processing power, while QoS focuses on user-perceived qualities, such as number of returned results and average query response time. In unstructured P2P systems, the quality of a search mechanism generally depends on the number of peers being explored (queried), response time, and traffic overhead. If more peers can be queried by a certain query, it is more likely that the requested object can be found. Here we define four performance metrics: average traffic cost versus search scope, query success rate, average neighbor distance, and query response time.

*Traffic cost* is one of the parameters network administrators are seriously concerned with. Heavy network traffic limits the scalability of P2P networks [70] and is also a rea-

son why a network administrator may prohibit P2P applications. We define the traffic cost as network resource used in an information search process of P2P systems, which is mainly a function of consumed network bandwidth and other related expenses. Specifically, in this work, we assume all the messages have the same length, so when messages traverse an overlay connection during the given time period, the traffic cost (C) is given by:  $C=M \times L$ , where M is the number of messages that traverse the overlay connection, and L represents the number of physical links in this overlay connection. *Search scope* is defined as the number of peers that queries have reached in an information search process. Thus, with the same traffic cost, we aim to maximize the search scope; while with the same search scope, we aim to minimize the traffic cost.

*Average neighbor distance (AD)* is used to evaluate the optimization results of a logical topology. Let  $AD_i$  be the average delay between the source peer  $i$  and all its logical neighbors. The value  $AD$  is defined as the average of all  $AD_i$ s (i.e., all peers in the P2P network). Minimizing average neighbor distance implies a better matching with the underlying physical network.

*Query success rate* is often an important metric in evaluating search efficiency and service quality. It measures the ratio of the queries for which at least one location of the desired data is found. If we use  $q_w(t)$  to denote the total number of queries issued by all the peers during the period from  $t-1^{th}$  to  $t^{th}$  time unit, and we use  $q_s(t)$  to denote the total number of queries for which one or more locations of the desired data are found, the query success rate at any given time  $t$ ,  $S(t)$ , is given by:

$$S(t) = \frac{q_s(t)}{q_w(t)} \times 100\%$$

Query success rate in an unstructured P2P system is dependent on many factors, such as the distribution of sharing contents, peers' dynamicity, initial values of query messages, search mechanism, etc. How to improve query success rate is beyond the discussion of this paper. When a P2P system is under DDoS attack, as we have discussed, delivery of query and query hit messages may fail due to blocked links and overloaded peers. Consequently, the average query success rate will be decreased.

*Response time* of a query is one of the parameters concerned by P2P users. We define response time of a query as the time period from when the query is issued until when the source peer received a response result from the first responder.

### **3.2 Simulation Methodology**

To evaluate effectiveness of our proposed approaches, we first generate network topologies. Based on generated networks, we simulate P2P flooding search, host joining/leaving behavior, and our proposed optimization operations.

Two types of topologies, physical topology and logical topology, are generated in our simulations. The physical topology should represent the real topology with Internet characteristics. The logical topology represents the overlay P2P topology built on top of the physical topology. All P2P nodes are in a subset of nodes in the physical topology. The communication cost between two logical neighbors is calculated based on the physical shortest path between this pair of nodes. To simulate the performance of different search mechanisms in a more realistic environment, the two topologies must accurately reflect the topological properties of real networks in each layer.

Previous studies have shown that both large scale Internet physical topologies [86] and P2P overlay topologies [72] follow the small world and power law properties. Power law

describes the node degree while the small world describes characteristics of path length and clustering coefficient [21]. The study in [72] found that the topologies generated using the AS Model have the properties of the small world and power law. BRITE [1] is a topology generation tool that provides the option to generate topologies based on the AS Model. Using BRITE, we generate physical topologies with 20,000 to 28,000 nodes. The logical topologies are generated with the number of peers (nodes) ranging from 2,000 to 8,000. The average number of neighbors of each node is ranging from 4 to 10.

### 3.2.1 Flooding search simulation

Our simulation is based on observed distributions as follows. Content popularity of a publisher follows Zipf-like distribution (aka Power Law) [16, 20], where the relative probability of a request for the  $i$ th most popular page is proportional to  $1/i^\alpha$ , with  $\alpha$  typically taking on some value less than unity. The observed value of the exponent varies from trace to trace. The request distribution does not follow the strict Zipf's law (for which  $\alpha=1$ ), but instead does follow a more general Zipf-like distribution. Query word frequency does not follow a Zipf distribution [41, 89]. User's query lexicon size does not follow a Zipf distribution [89] but with a heavy tail. Both the overall traffic and the traffic from 10% popular nodes are heavy-tailed in terms of the host connectivity, traffic volume, and average bandwidth of the hosts [78]. Studies in [76] have suggested a log-quadratic distribution ( $10^{-\alpha^2}$ ) for stored file locality and transfer file locality. The time length that nodes remain available follows a log-quadratic curve [76], which could be approximated by two Zipf distributions.

In our simulation, we simulate flooding search used in Gnutella and KaZaA networks by conducting the Breath First Search (BFS) algorithm from a specific node. A search operation is simulated by randomly choosing a peer as the sender, and a keyword according to Zipf distribution. In our first simulation, more than 1,000,000 search operations are simulated sequentially.

### **3.2.2 A dynamic P2P environment**

P2P networks are highly dynamic with peers joining and leaving frequently. The observations in [78] have shown that over 20% of the logical connections in a P2P last one minute or less, and around 60% of the IP addresses keep active in FastTrack for no more than 10 minutes each time after they join the system. The measurement reported in [72] indicated that the median up-time for a node in Gnutella and Napster is 60 minutes. Studies in [18] have argued that measurement according to host IP addresses underestimates peer-to-peer host availability and have shown that each host joins and leaves a P2P system 6.4 times a day on average, and over 20% of the hosts arrive and depart every day. Although the numbers they provided are different to some extent, they share the same point that the peer population is quite transient. We simulate the joining and leaving behavior of peers via turning on/off logical peers. In our simulation, every node issues 0.3 queries per minute, which is calculated from the observation data shown in [82], i.e., 12,805 unique IP addresses issued 1,146,782 queries in 5 hours. When a peer joins, a lifetime in seconds will be assigned to the peer. The lifetime of a peer is defined as the time period the peer will stay in the system. The lifetime is generated according to the distribution observed in [72]. The mean of the distribution is chosen to be 10 minutes [78]. The value of the variance is chosen to be half of the value of the mean. The lifetime will be

decreased by one after passing each second. A peer will leave in next second when its lifetime reaches zero. During each second, there are a number of peers leaving the system. We then randomly pick up (turn on) the same number of peers from the physical network to join the overlay.

## 4 Overlay Optimization in P2P Systems

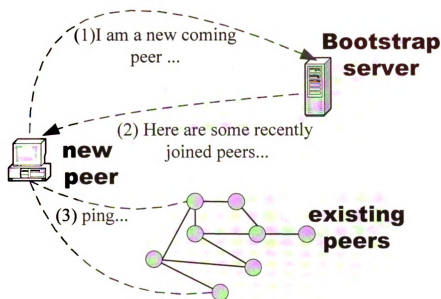
The inefficient P2P overlay topology is a major reason for the heavy P2P traffic [69]. In this chapter, we discuss unnecessary query message duplications on both overlay level and IP level. We define the Optimal Overlay Problem, and prove it is NP hard. We then discuss our proposed approaches to optimizing the P2P overlays and prove the effectiveness of these algorithms by simulation studies.

### 4.1 *Optimal Overlay Problem*

In a P2P system, all participating peers form a P2P network over a physical network. A P2P network is an abstract, logical network called an overlay network. Maintaining and searching operations of a Gnutella peer are specifically described in [9]. When a new peer wants to join a P2P network, a bootstrapping node provides the IP addresses of a list of existing peers in the P2P network. The new peer then tries to connect with some of these peers. If some attempts succeed, the connected peers will be the new peer's neighbors. Figure 4.1 illustrates a typical process of bootstrapping a new peer in Gnutella.

Once this peer connects into a P2P network, the new peer will periodically ping the network connections and obtain the IP addresses of some other peers in the network. These IP addresses are cached by this new peer. When a peer leaves the P2P network and

wants to join the P2P network again (no longer the first time), the peer will try to connect to the peers whose IP addresses have already been cached.



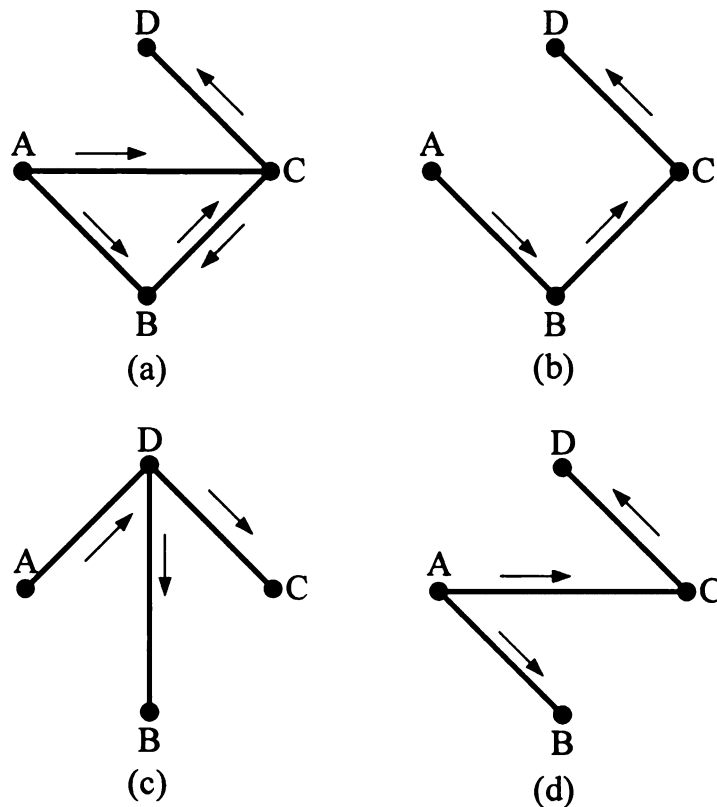
**Figure 4.1 Bootstrapping a new peer in Gnutella**

#### **4.1.1 Message Duplications in Overlay Connections**

Figure 4.2 shows some examples of P2P overlay topologies where solid lines denote overlay connections among logical P2P neighbors. Consider the case when node A issues a query. A solid arrow represents a delivery of the query message along one logical connection. In Gnutella, a peer forwards an incoming query message to all of its directly connected peers, except the one that delivered the incoming query. Thus, as shown in Figure 4.2 (a), A's query is relayed by nodes B and C. Peer B forwards the query to C, while C also forwards the query to B. In this case, the pair of transmission between B and C is unnecessary message duplication. We can easily observe that the other three overlays

shown in Figure 4.2 (b)-(d) have less message duplications, while retaining the same search scope for this query.

However, we cannot draw the conclusion that the overlays in Figure 4.2 (b)-(d) are better than the one in Figure 4.2 (a) because the above discussion only takes traffic cost into consideration. In fact, compared with Figure 4.2 (a), the overlays in Figure 4.2 (b)-(d) have less overlay connections, but may cause longer average query response times. For example, when A issues larger amount of queries and D has most of the desired data, the query response time in the overlay in Figure 4.2(b) will be much longer than that in other three overlays.

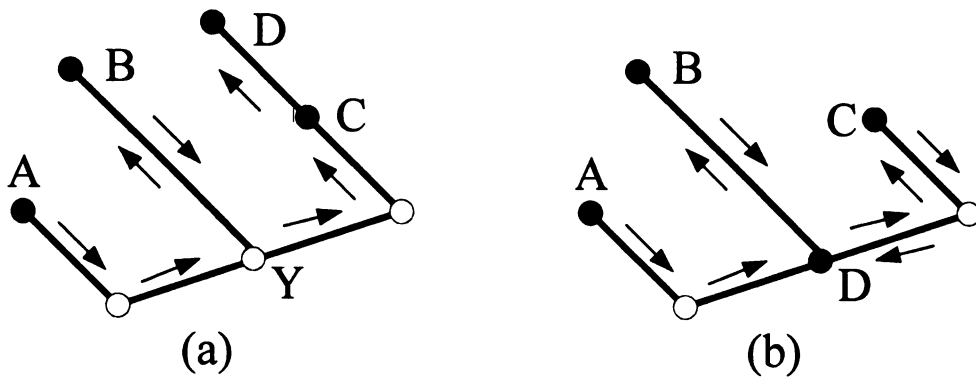


**Figure 4.2 Examples of P2P overlay topologies**

Generally, as long as cycles exist in search paths, there must be message duplications in overlay connections. Some peers, such as B and C, are visited by the same query message multiple times. If a peer receives a query message with the same Message ID (GUID) as the one it has received before, the peer will discard the message. Since a peer is aware of this kind of revisit, we call it a Revisit Known (RK) problem. The price of reducing RK duplications is the increment of query latency. Our first motivation is to reduce message duplications in overlay level and attack RK problems with minimal increment of query response time, while retaining the same search scope of queries.

#### 4.1.2 Message Duplications in Physical Links and Topology Mismatch Problem

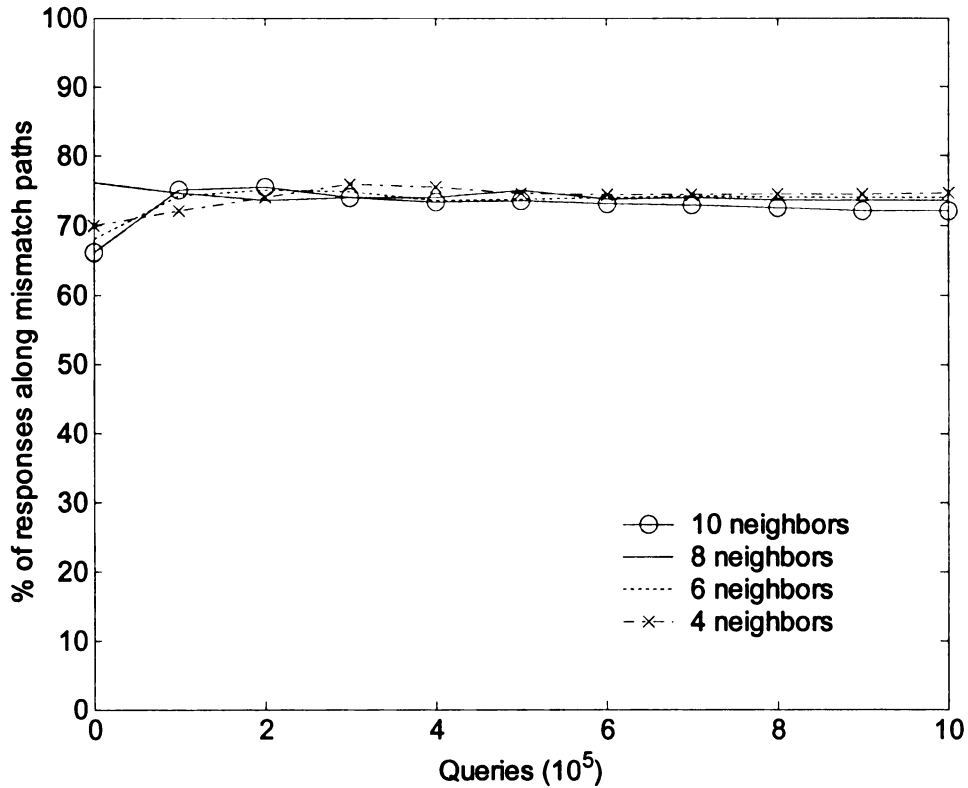
We have discussed message duplications in overlay connections. However, for an overlay without RK problem, the same message still can traverse the same physical link multiple times, causing large amount of unnecessary traffic and increasing query response time. Here is an example. Suppose Figure 4.3 (a) illustrates the underlying physical network of the overlay shown in Figure 4.2 (b), where A, B, C and D are peering nodes and node Y is not a peering node. We can see that the query message along the overlay path  $A \rightarrow B \rightarrow C \rightarrow D$  traverses physical link YB twice. Node Y is visited twice.



**Figure 4.3 Examples of physical topologies**

Since node Y is not a peering node, the message duplication (revisit to Y) cannot be avoided. We may reduce the duplication between link YB by creating a direct connection between A and C, and disconnecting the logical link BC, as shown in Figure 4.2 (d), but new duplications may occur in other links, such as YA.

Figure 4.3 (b) shows another underlying physical network of the overlay shown in Figure 4.2 (b). For a query message along the overlay path  $A \rightarrow B \rightarrow C \rightarrow D$ , D is visited three times. Node D is a peering node, but in the first two visits, D is visited as a non-peering node. These first two visits are not known by the P2P application. We call this kind of revisits as a Revisit Not known (RN) problem. In this case, three physical links have been traverses twice, as shown in Figure 4.3 (b), a topology mismatch problem occurs.



**Figure 4.4 Percentage of query responses along mismatched paths**

It is more effective to solve RN problems than RK problems since RN problems will not only increase message duplications/traffic cost as RK problems, but also increase query response time. In Figure 4.3 (b), node D has been visited by the same query message twice before it ‘formally’ receives the query as a peering node. If we can replace the overlay in Figure 4.2 (b) by the one in Figure 4.2 (c) for physical topology in Figure 4.3 (b), there will be no message duplications at all, and the response time from D to A will be decreased significantly. Our second motivation is to improve search performance by alleviating RN problems.

In fact, the stochastic peer connection and peers’ randomly joining and leaving a P2P network can cause large amount of topology mismatch between the P2P logical overlay network and the physical underlying network. Studies in [69] have shown that only 2 to 5 percent of Gnutella connections link peers within a single autonomous system (AS). But more than 40 percent of all Gnutella peers are located within the top 10 ASes. This means that most Gnutella-generated traffic crosses AS borders so as to increase topology mismatching costs. Our simulation results in Figure 4.4 show that 744,734 out of 1,000,000 query responses traverse along mismatched paths, in each of which at least one of the peering nodes is visited as a non-peering node for more than once.

### **4.1.3 Modeling P2P Networks**

We model a P2P network based on the following assumptions. First, an overlay connection between a pair of peering nodes consists of a number of physical links which form a shortest path between the pair of end nodes in the physical topology, and Internet paths are relatively stable [95]. Second, we assume that the same size packets traversing

the same physical link in a short period of time will have similar delay, as assumed by many other measurement applications [33, 85].

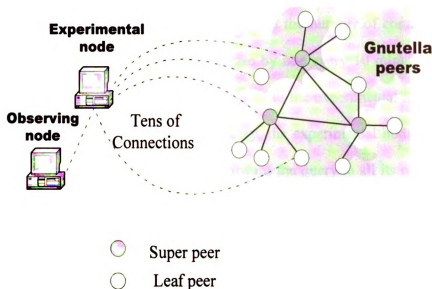
Graph theoretic terms not defined here can be found in [23]. We model a communication network by an undirected graph  $G = (V, E)$  where the vertex set  $V$  represents units such as hosts and routers, and the edge set  $E$  represents physical links connecting pairs of communicating unit. For instance,  $G$  could model the whole or part of the Internet.

Given an undirected graph  $G=(V, E)$  modeling an interconnection network, and a subset  $X \subseteq V(G)$  of communicating units (peers), we construct a corresponding complete edge weighted graph  $D=(V, E)$ , where  $V(D) = X$ , and the weight of each  $uv \in E(D)$  is equal to the length of a shortest path between peer  $u$  and peer  $v$  in  $G$ . Note that  $D$  is a complete graph, that is, it includes all possible edges, and is referred to as the distance graph of  $G$ .

In the context of our discussion, we start with a physical network  $G$  (perhaps representing the Internet), and then choose a set of communicating peers  $X$ . The resulting distance graph  $D$ , constructed as mentioned earlier, is the basis for constructing a P2P overlay graph  $H=(V, E)$ , which is done as follows. The vertex set  $V(H)$  will be the same as  $V(D)$ , and edge set  $E(H) \subseteq E(D)$ . The key issue here is how to select  $E(H)$ . For the remainder of the paper, we will consistently refer to the “physical” graph by  $G$ , its distance graph by  $D$ , and an overlay graph corresponding to  $D$  by  $H$ .

From the process of building a P2P network, as we have previously discussed, theoretically,  $E(H)$  could be any subset of  $E(D)$ . However, an optimal overlay network should have the following basic properties. First, the selection of  $E(H)$  should make  $H$  a connected graph. We define search scope as the number of peers that a query can reach in an

information search process. Although in real systems,  $H$  could consist of several isolated components, an optimal overlay should include only one component such that a query can reach all peering nodes if the TTL is large enough.



**Figure 4.5 A super peer behaviour observation experiment**

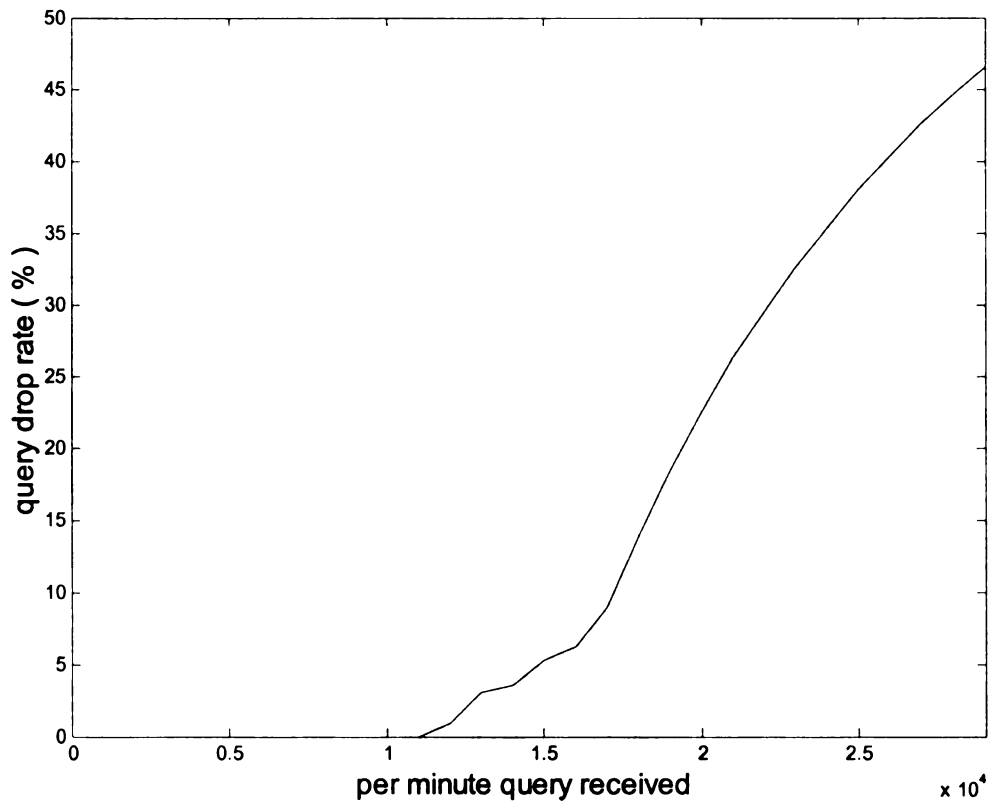
Second, the overly network  $H$  should be degree-bounded to balance the load of peers since an extremely large connection degree (the number of peering neighbors) of a peer causes heavy loading and query dropping, and an extremely small degree of a peer causes very long response times. We identify that the upper bound of a peer's degree for the peer to not become a query bottleneck is 8, by experiments described as below.

As shown in Figure 4.5, we build an experimental node to receive and forward queries flooding through the Gnutella network, and an observing node that receives queries from the experimental node but does not process or issue any queries. Using a modified LimeWire [11] client with logging functionality, all the queries passing by the experi-

mental and observing nodes are recorded to two log files. Each of the two nodes was running on a PC with a 2.4GHz Pentium IV processor and 100M Ethernet interface. This experiment observing a super peer's behavior lasted 24 hours.

During the experiment, the experimental node, as shown in Figure 4.5, is intentionally configured as a super peer connecting to tens of peers in the Gnutella network, including super peers and leaf peers. We tightly control the number of connections to the experimental node, and let this number increase by one every 10 minutes. As the number of connections increases, more queries are sent to the experimental node. According to the Gnutella protocol, for each received query, the experimental node will first look up its local sharing storage index, and then forward the query to all its neighbors including the observing node but excluding the query incoming peer. The observing node counts the number of queries forwarded by the experimental node so as to measure the capability of a peer in processing search queries and observe a super node's behavior. We show the result in Figure 4.6, where the experimental node starts dropping query messages when there are more than 11k queries coming per minute.

There is no close correlation between the number of connections and the amount of incoming queries, but through the experiment we find the increase of the number of queries is trivial when one more leaf node is connected compared with having one more super peer neighbor. This is because a super peer can send out thousands of queries while a leaf peer only sends out one query in several minutes. When there are more than 6 to 8 super peer neighbors, the experimental node sometimes witnesses more than 11k incoming queries per minute. When the peer has more than 12 super peer neighbors, there are always more than 11k incoming queries per minute.



**Figure 4.6 Query drop rate vs. query density**

Note that in our experiment the experimental node is dedicated to this experiment, while in a real system a peer may have other conventional tasks. Furthermore, normally a peer's local index includes many contents; while in our experiment the local index is almost empty, which will reduce the time for local look-up operations. Based on these observations, we bound a peer's degree by 8, ignoring the connections with leaf peers in super peer systems since they are not involved in query forwarding processes but only issuing queries and waiting for responses.

#### **4.1.4 Amount of Message Duplications in Overlay Connections**

Ther

to pure

lations

lay lin

Ge

overl

a pee

ceive

**T**

arbi

dup

**I**

im

ca

tr

th

gr

als

tra

There is a tradeoff between traffic cost and query response time [97]. It is meaningless to purely optimize one metric without considering the other. We need to establish the relationship between message duplications in overlay connections and the number of overlay links.

Generally, as long as loops exist in search paths, there must be message duplications in overlay connections. Some peers are visited by the same query message multiple times. If a peer receives a query message with the same Message ID (GUID) as the one it has received before, the peer will discard the message.

**Theorem 1:** Let  $H(V, E)$  be an overlay graph. An arbitrary query message issued by an arbitrary peer with a sufficiently large TTL value can result in  $d = 2(|E(H)| - |V(H)| + 1)$  duplicate messages in the overlay graph.

**Proof:** For obvious reasons, we will assume that  $H$  is nontrivial and connected, which implies  $H$  will contain at least  $|V(H)| - 1$  edges. The proof is done by induction of  $E(H)$ .

**Basis:**  $|E(H)| = |E(V)| - 1$

In this case  $H$  is a tree, and therefore there is no loop in the graph to generate duplicated message, and thus the assertion holds, as  $d = 2(|E(H)| - |V(H)| + 1) = 0$ .

**Hypothesis:** Assume the assertion is correct for  $|E(H)| < k$ , where  $k \geq |V(H)|$ .

**Step:** We want to show that the assertion is correct when  $|E(H)| = k$ . Let  $H$  be an arbitrary overlay connected graph with  $|E(H)| = k$  edges. Since  $H$  has more than  $|V(H)|$  edges, then  $H$  must contain at least one cycle  $C$ . Let  $uv$  be an edge of  $C$ , and now consider the graph  $H' = H - uv$ . Since  $H$  was assumed to be connected, and  $uv$  belongs to a loop,  $H'$  is also connected. Further, since  $|E(H')| < k$ , then by induction hypothesis, there is an arbitrary query message, call it  $Q$ , issued by an arbitrary peer with a sufficiently large TTL

value that results in  $2(|E(H')| - |V(H')| + 1)$  duplicate messages in the overlay graph  $H'$ . Now if we initiate the same query  $Q$  in  $H$ , sticking to the same message propagation as in  $H'$ , at some point peers  $u$  and  $v$ , the end vertices of the edge  $uv$ , will be informed. These peers in turn can potentially send messages to each other, generating two unnecessary messages on the link  $uv$ . Therefore, the number of duplications in  $H$  for the message query  $Q$  will be  $2(|E(H')| - |V(H')| + 1) + 2$ . Rewriting this expression in terms of  $E(H)$  and  $V(H)$ , and remembering that  $V(H) = V(H')$ , we get  $2(|E(H')| - |V(H')| + 1) + 2 = 2(|E(H')| + 1 - |V(H')| + 1) = 2(|E(H)| - |V(H)| + 1)$ , and thus the theorem. ■

More message duplications in overlay links means more traffic cost. In an overlay  $H=(V, E)$  with  $p$  nodes, concerning the traffic cost only, the best case is when  $H=(V, E)$  is a spanning tree reduced from the complete distance graph  $D=(V, E)$ . In this case, a query incurs  $n-1$  messages in overlay connections to reach all peers. However, the average query response time will be long compared with a graph with more overlay connections. Indeed, the number of overlay connections balances the traffic cost and average query response time. As it is impossible to minimize these two metrics simultaneously, we seek an optimal overlay when the cardinality of  $E$  is given.

#### 4.1.5 Optimal Overlay Problem

**Definition 4.1:** Let  $F=(V, E)$  be an edge weighted connected graph. Further, let  $dis(u, v)$  represent the distance (that is, the length of a shortest path) between vertex  $u$  and  $v$ . The average distance, denoted by  $AD(F)$ , of graph  $F$  is defined as follows:

$$AD(F) = \frac{1}{\binom{|V(F)|}{2}} \sum_{\text{unordered pair } u, v \in V(F)} dis(u, v)$$

We now define the Degree-bounded Minimum Average Distance (DMAD) overlay problem.

**Definition 4.2:** Let graph  $G=(V, E)$  represent a physical network and let graph  $D = (V, E)$  be its distance graph for a set of communicating peers, as defined earlier. Recall that  $D$  is a complete graph. Furthermore, let  $k$  be an integer such that  $\delta(G) \leq k \leq \Delta(G)$ , where  $\delta(G)$  and  $\Delta(G)$  are the minimum and the maximum degree of  $G$ , respectively. A DMAD overlay graph  $H= (V, E)$  is a connected spanning subgraph of  $D$  having the following properties:

- (a)  $\Delta(H) \leq k$
- (b)  $AD(H)$  is as small as possible, subject to (a).

The DMAD problem is a generalization of the degree-bounded connected subgraph problem (DBCS) which asks the following question [37]:

**Instance:** Graph  $G=(V, E)$ , non-negative integer  $d \leq |V(G)|$ , positive integer  $k \leq |E(G)|$ .

**Question:** Is there a subset  $E' \subseteq E(G)$  with  $|E'| \geq k$  such that the subgraph  $G'=(V, E')$  is connected and has no vertex with degree exceeding  $d$ ?

Clearly there is a straightforward polynomial transformation from the DBCS problem to the decision version of the DMAD problem, and thus DMAD problem is NP-hard. ■

Knowing that DMAD problem is NP-hard leaves little hope for finding an optimal overlay network even when we have complete information about the overlay network. Worse yet, it is practically impossible for a peer to collect global knowledge of the overlay topology since the number of online users could be millions and these P2P users are randomly coming and leaving. However, it is clear that optimizing inefficient overlay to-

pologies can fundamentally improve P2P search efficiency. Thus, we intend to develop several overlay optimization algorithms which have the following properties. First, they must be completely distributed and do not need any global knowledge of the overlay or the underlying physical topology. Second, the traffic overhead incurred by the algorithms and computation overhead should be trivial compared with the traffic cost savings. Third, as the P2P users are randomly coming and leaving, the convergent speed of the algorithms must be fast enough so that it is effective in dynamic environments. Finally, the search scope is not shrunk by the optimization operations.

#### ***4.2 Adaptive Overlay Topology Optimization (AOTO)***

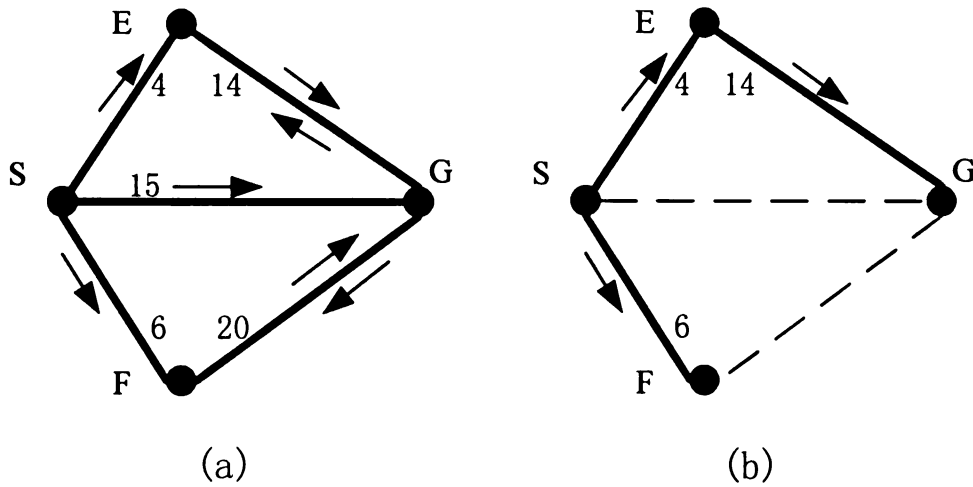
If the system can detect and disconnect the slow logical connections and alleviate topology mismatch (RN) problems, the total network traffic could be significantly reduced without shrinking the search scope of queries. This is the basic principle of our proposed Adaptive Overlay Topology Optimization [53] (AOTO) to address the topology mismatch problem. While retaining the desired prevailing unstructured architecture of P2P systems, the goal of AOTO is to dynamically optimize the logical topology to improve the overall performance of P2P systems. AOTO includes two steps: Selective Flooding (SF) and Active Topology (AT). Selective Flooding is to build an overlay multicast tree among each peer and its immediate logical neighbors, and route messages on the tree to reduce flooding traffic without shrinking the search coverage range. Thus, some neighbors become non-flooding neighbors. Active Topology is the second step in AOTO for each peer to independently make optimization on the overlay topology to alleviate topology mismatch problem by replacing non-flooding neighbors with closer nodes as direct logical neighbors.

### 4.2.1 Selective Flooding

Instead of flooding to all neighbors, SF uses a more efficient flooding strategy to selectively flood a query on an overlay multicast tree. This tree can be formed using a minimum spanning tree algorithm among each peer and its immediate logical neighbors. In order to build the minimum spanning tree, a peer has to know the costs to all its logical neighbors and the costs between any pair of the neighbors. We use network delay between two nodes as a metric for measuring the cost between nodes. We modify the Limewire implementation of Gnutella 0.6 P2P protocol by adding one routing message type. Each peer probes the costs with its immediate logical neighbors and forms a routing message type.

The peer then forms a neighbor cost table. Two neighboring peers exchange their neighbor cost tables so that a peer can obtain the cost between any pair of its logical neighbors. Thus, a small overlay topology of a source peer and all its logical neighbors is known to the source peer. Based on obtained neighbor cost tables, a minimum spanning tree then can be built by simply using an algorithm like PRIM which has a computation complexity of  $O(m^2)$ . Now the message routing strategy of a peer is to select the peers that are the direct neighbors in the multicast tree to send its queries. An example is shown in Figure 4.7. In Figure 4.7(a), the traffic incurred by node S's flooding of messages to its direct neighbor E, F, and G is:  $4+14+14+15+6+20+20=93$ .

After SF computing, we can see the forwarding connections are changed as shown in Figure 4.7(b), and the total traffic cost becomes:  $6+4+14=24$ .



**Figure 4.7 Selective Flooding**

In Figure 4.7(b), node S sends a message only to nodes E and F and expects that node E will forward the message to node G. Note that in this step, even node S does not flood its query message to node G any more. S still retains the connections with G and keeps exchanging the neighbor cost tables. We call node G *non-flooding neighbor* of node S, which is the direct neighbor potentially to be replaced in the next step.

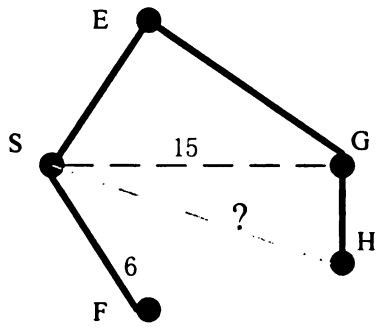
#### 4.2.2 Active Topology

The second step of AOTO, AT, reorganizes the overlay topology. Note that each peer has a neighbor list which is further divided into flooding neighbors and non-flooding neighbors in SF. Each peer also has the neighbor cost tables of all its neighbors. In this step, it tries to replace those physically far away neighbors by physically close by neighbors, thus minimizing the unnecessary traffic caused by topology mismatch.

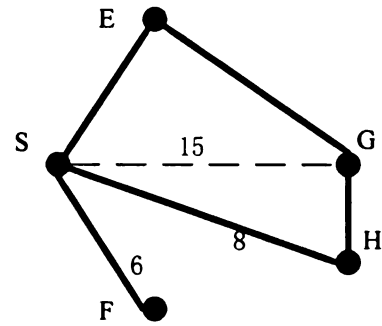
An efficient method to identify such a candidate peer to replace a far away neighbor is critical to the system performance. Many methods may be proposed. In AOTO, a non-flooding neighbor may be replaced by one of the non-flooding neighbor's neighbor.

The basic concept of AT is illustrated in Figure 4.8. In this example, node S tries to probe the distance to one of its non-flooding neighbor G's neighbors, peer H. If SH is smaller than SG, connection SG will be disconnected. If SG is smaller than SH, but S finds that the cost between nodes G and H is even larger than the cost between nodes S and H, S will keep H as a new neighbor. Since the algorithm is executed in each peer independently, S cannot let G to remove H from its neighbor list. However, as long as S keeps both G and H as its logical neighbors, we may expect that node H will become a non-flooding neighbor to node G after node G's SF step since node G expects S to forward messages to H to reduce unnecessary traffic. Then G will try to find another peer to replace H as its neighbor. After knowing that H is no longer a neighbor to G from periodically exchanged cost tables from node G (or from node H), S will cut the connection SG, although S has already stopped sending query messages to G for a period of time since the spanning tree was built for S. Obviously if SH is larger than SG and GH, this connection will not be built and S will keep probing other G's director neighbors.

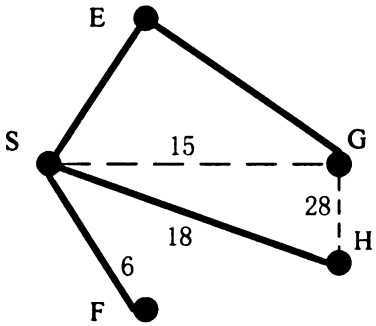
It is very important to quickly identify the best candidate from a non-flooding neighbor's neighbor list to minimize replacement overhead. The ideal case would be that we can always choose the candidate with lowest cost to the source at the first time.



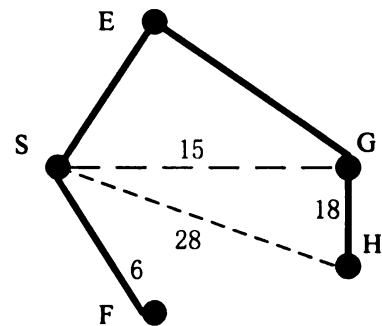
(a) S probes G's neighbor H



(b)  $SH < SG$ , replace G by H



(c)  $SH > SG$ , but  $SH < GH$ , S keeps H as a direct neighbor



(d)  $SH > SG$  and  $SH > GH$ , S starts probing next G' neighbor

**Figure 4.8 Active Topology**

In our simulation, we use three different policies to choose the candidate. The naïve policy, which is not based on SF optimization, simply disconnects the source node's most expensive neighbor. The source node will probe the costs to some other nodes, and try to find a less expensive node as a replacement of the disconnected neighbor. The naïve policy cannot guarantee the same search scope as in original topology. The replacement strategy is also not efficient. The other two policies are based on SF optimization. The first one is random policy in which the source randomly picks a node from the source's non-flooding neighbor's neighbor list. The source then decides if the selected candidate

will be selected. The second one is closest policy in which the source will probe the costs to all of the non-flooding neighbor's neighbors, and select the closest one.

Let  $C_{ij}$  represent the cost from peer  $i$  to peer  $j$ . The following pseudo code describes the randomized AT algorithm for a given source peer  $i$ .

#### **Pseudo Code of the Randomized AT Algorithm (peer $i$ )**

**For** each  $j$  in  $i$ 's non-flooding neighbors

    Replaced = false;

    List = all  $j$ 's neighbors excluding  $i$ ;

**While** List is not empty and Replaced = false

        randomly remove a peer  $h$  from List;

        measure  $C_{ih}$ ;

**if**  $C_{ih} < C_{ij}$  {replace  $j$  by  $h$  in  $i$ 's neighbor list;

            Replaced = true}

**else if**  $C_{ih} < C_{jh}$  { add  $h$  to  $i$ 's neighbor list;

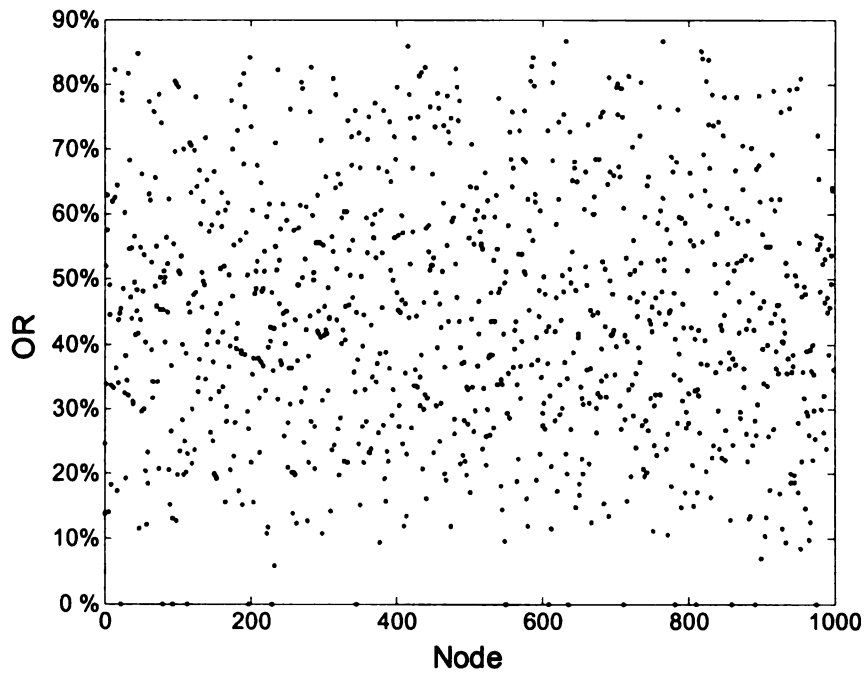
            Replaced = true};

**End While;**

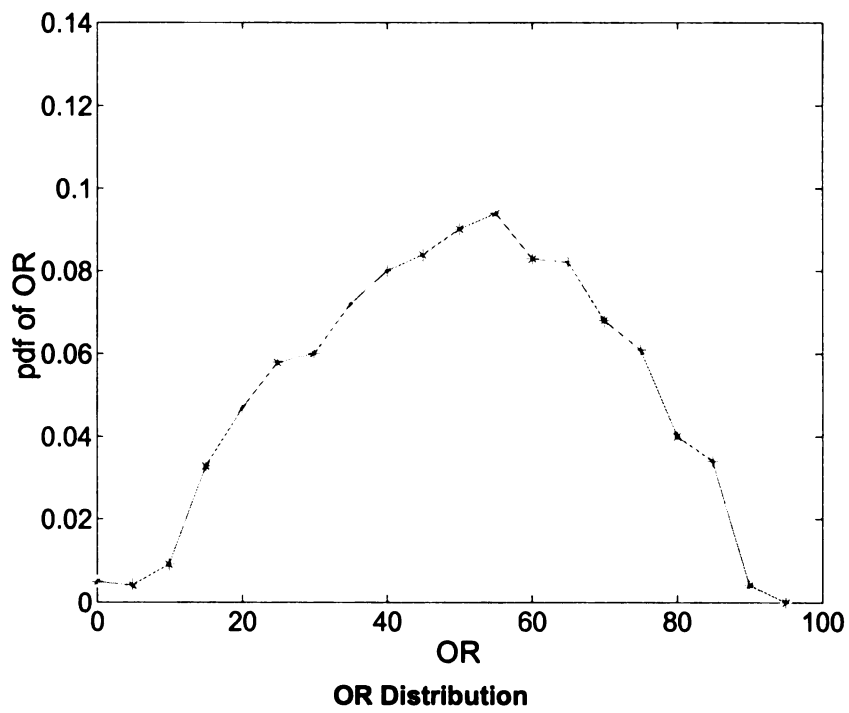
**End For;**

#### **4.2.3 Effectiveness of Select Flooding Procedure**

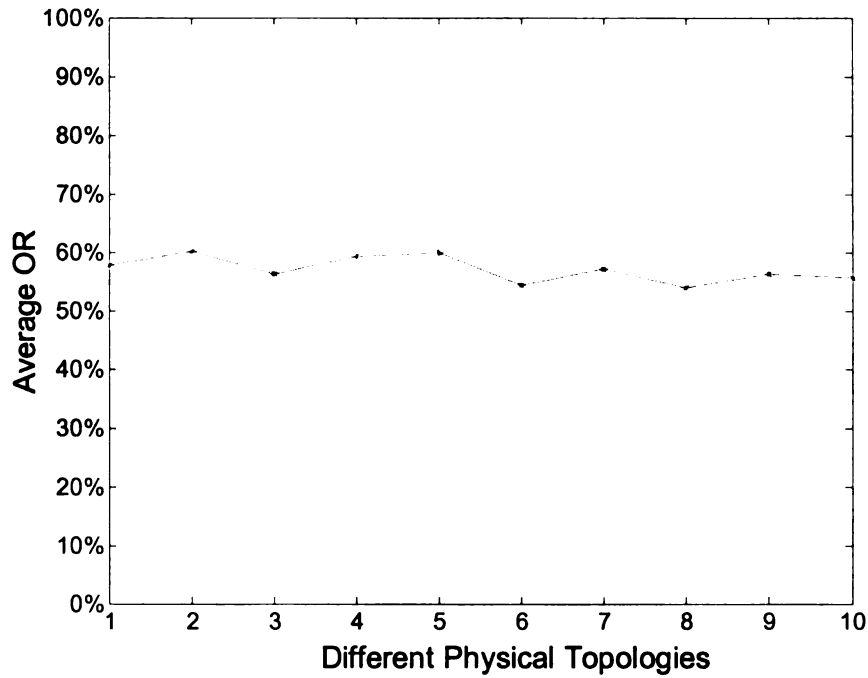
In our first simulation, a 1000 node logical graph with an average 8 edges connections (average 16 logical neighbors) is used. We define Optimization Rate (OR) as the ratio of the traffic cost before and after SF operations. Figure 4.9 shows the different OR of each node after their first step of SF. More than 90% of the nodes will reduce the cost when using the SF strategy to make their queries reach all logical neighbors. The average optimization rate is around 40-50%. After this first simulation, we wonder what is main factor that influences the average OR because OR shows the effectiveness of SF and will decide whether the peer will enter the next step of AT.



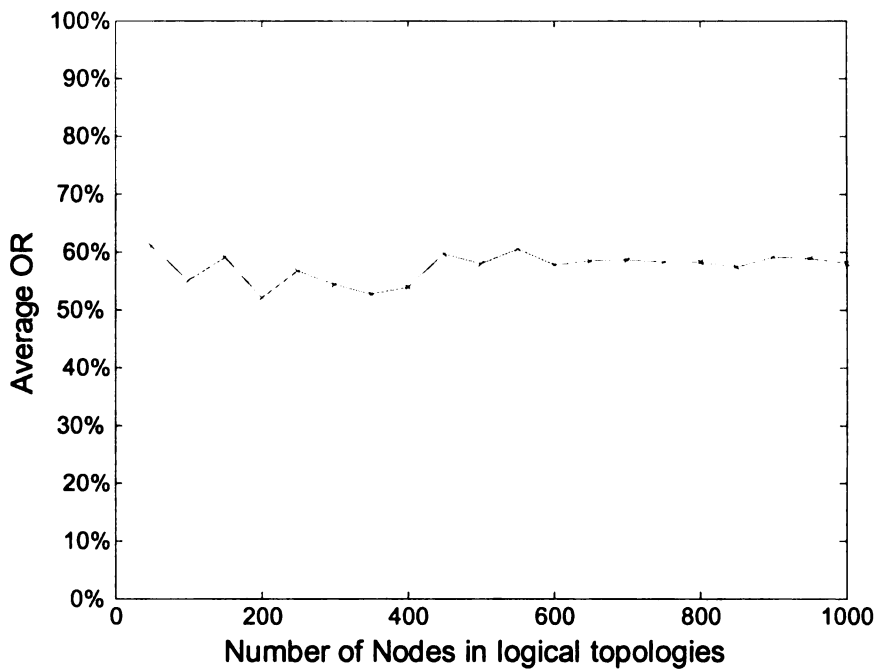
**ORs of 1000 nodes**



**Figure 4.9 ORs and OR distribution on a 1000-node logical topology**



**Figure 4.10 Average ORs on 10 physical topologies**



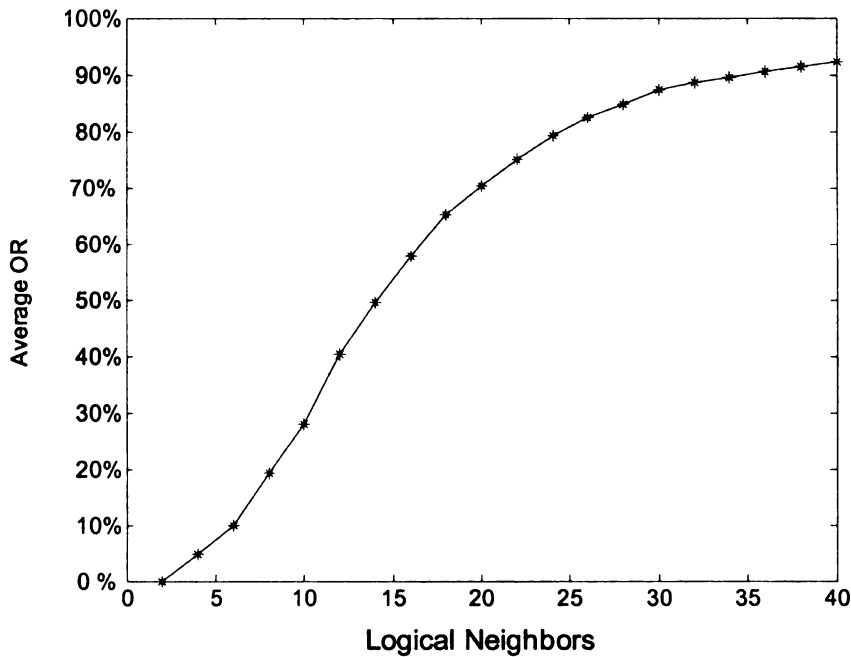
**Figure 4.11 Average ORs on 20 logical topologies with different number of nodes**

After we tried the same logical graph based on 10 different physical topologies, we obtained consistent results, which are shown in Figure 4.10.

We then simulated AOTO on different logical topologies with different number of nodes ranging from 50 to 1000. The results in Figure 4.11 show that the density of P2P nodes does not influence the effectiveness of SF. They all have an average optimization rate at around 50%.

When we changed the number of edge connections in the logical topology, the optimization rate changes greatly. Figure 4.12 shows the performance of 20 500-node logical topologies with different average logical connections ranging from 2 to 40.

The results show that SF is more effective with large number of logical neighbors. For example, SF can achieve the average optimization rate as high as 87.4% on a logical topology with an average of 30 logical neighbors.



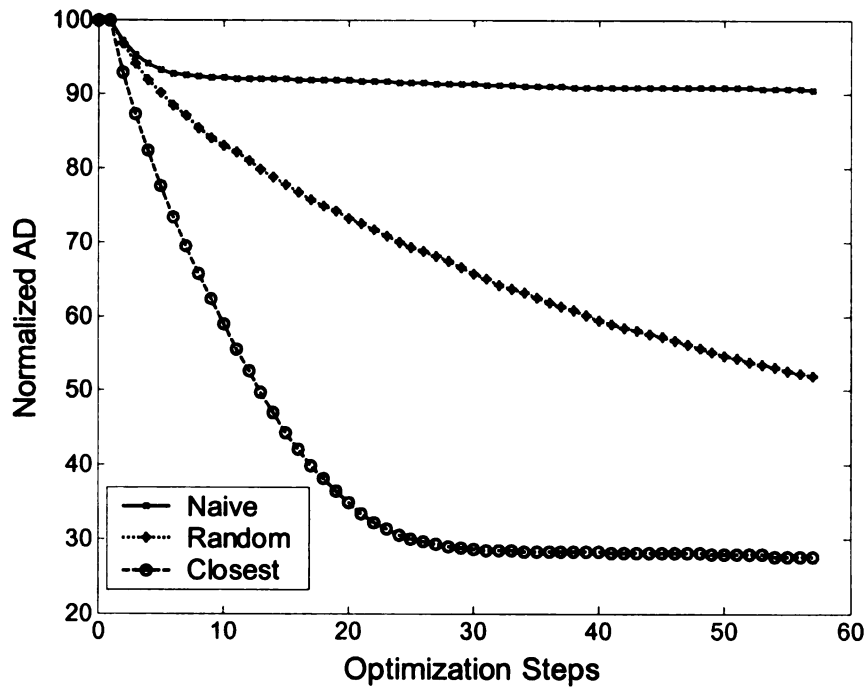
**Figure 4.12 OR for different average neighbor numbers**

#### 4.2.4 Effectiveness of Active Topology Procedure

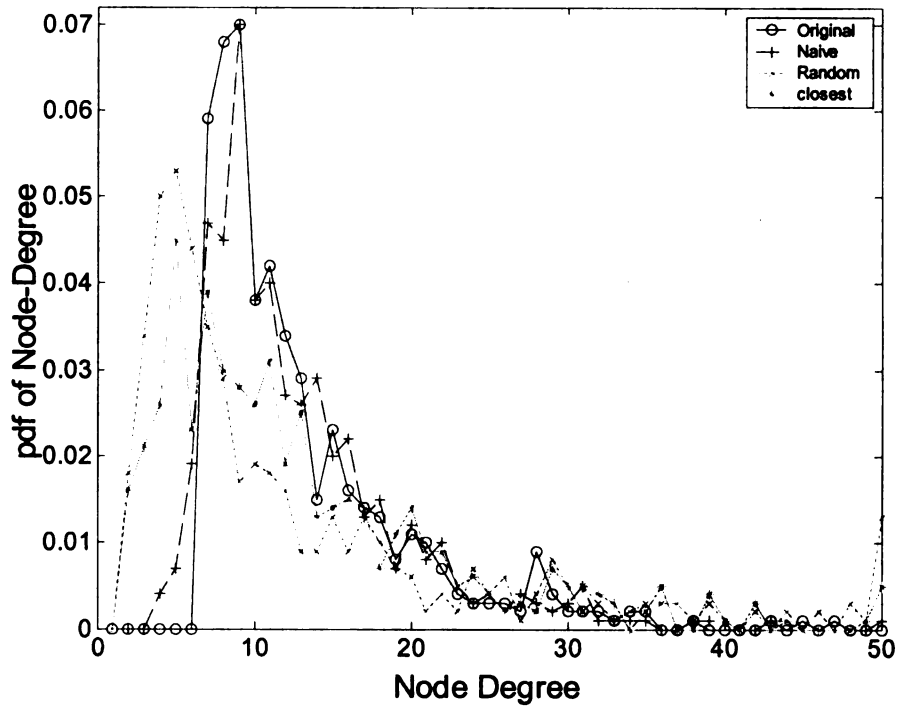
We evaluate the result of optimizing a logical topology by computing Average Neighbor Distance. After each node computes the optimization rate, the node enters the procedure of AT. As we discussed, AT is to optimize the logical topology by choosing closer neighbors, so as to attack the mismatch problem.

There are three different policies in AT to select candidate peers to replace non-flooding peers identified in SF. We compare the average distances that are normalized to 100 of the three policies, i.e. naïve, random and closest, as the optimization steps are increased from 0 to 59. Figure 4.13 shows that the naïve policy is least effective. More optimization steps can hardly produce lower AD. Both the random and closest policy work well, and closest is the most effective. However, the computation complexity of closest policy is  $O(mn)$ , while that of random policy is only  $O(n)$ , where  $m$  is the average number of logical neighbors (branching factor) and  $n$  is the total number of nodes in a P2P logical topology. Figure 4.14 plots node-degree's pdf distribution before and after AT optimizations. We can see that optimized logical topologies using different policies keep the similar branching factor property as the original logical topology so that the query search scope can be guaranteed.

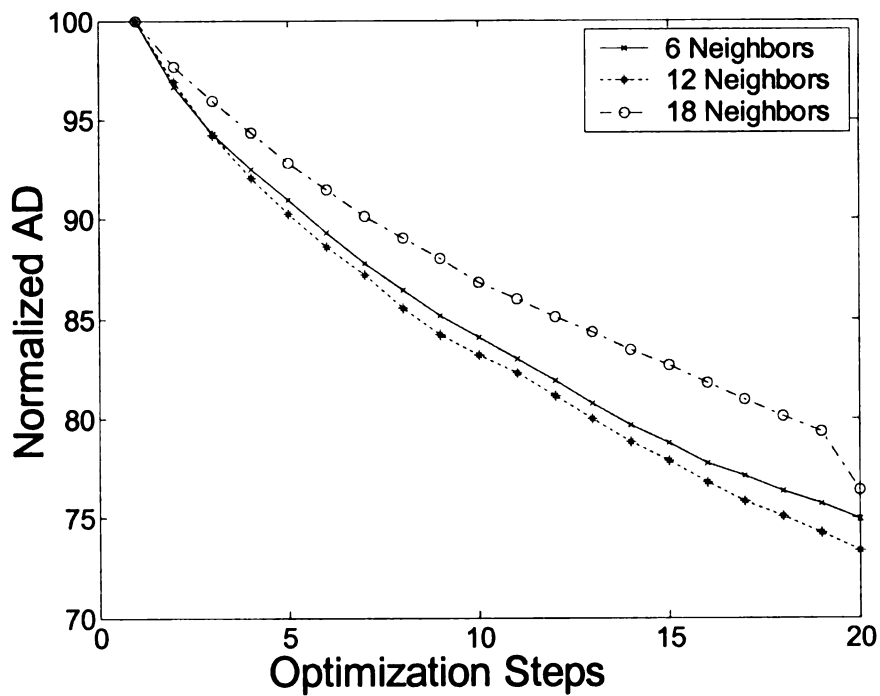
The average number of logical neighbors is a major factor to affect the effectiveness of SF. But it is not true in AT. We compare the average reductions of AD on three 500-node logical topologies with average 6, 12, and 18 edge connections using random policy as the optimization steps are increased. Results in Figure 4.15 show that the number of logical neighbors has little impact to the effectiveness of AT.



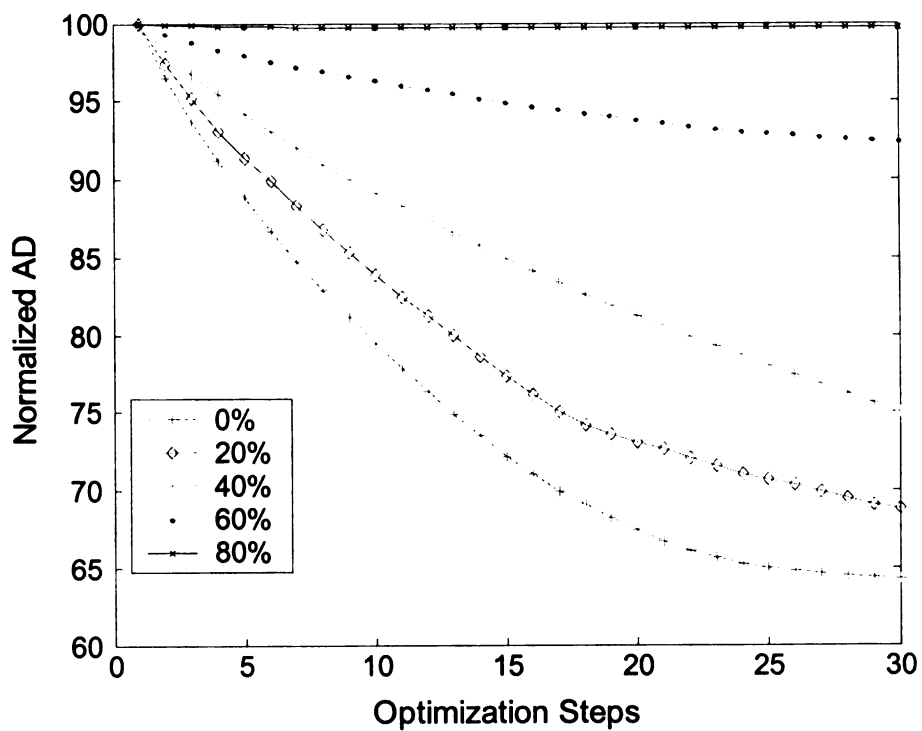
**Figure 4.13 Optimization for 3 Ways**



**Figure 4.14 Topology degree properties changes**

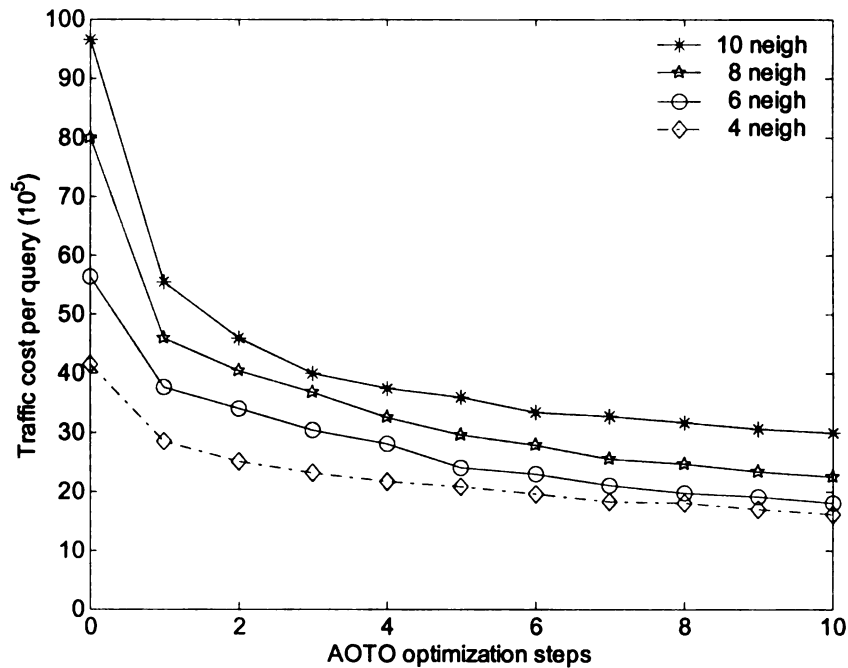


**Figure 4.15 Cost optimization**

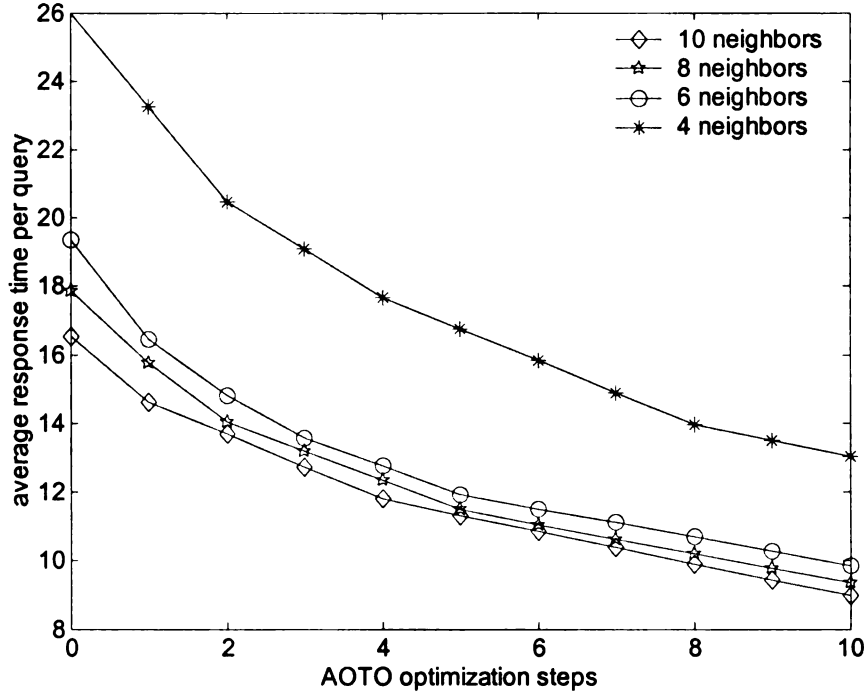


**Figure 4.16 Threshold factors**

In theory, each peer can continuously do SF and AT until no cost improvement is obtained, thus closing to a perfect topology matching. Obviously, this is unnecessary and creates too much overhead. In practice, after each replacement, the source peer will compute the cost improvement ratio and decide whether it needs to find another candidate peer to replace another non-flooding neighbor based on a termination threshold,  $\Delta$ . The optimization process will terminate if the improvement ratio is less than  $\Delta$ . Thus, the value of  $\Delta$  is a factor to impact the effectiveness of AT. A smaller threshold causes larger overhead. Figure 4.16 plots the normalized AD on different thresholds as the optimization steps are increased. AD is reduced slower for a larger threshold, and a lower threshold leads to a better result.



**Figure 4.17 Traffic reduction vs. optimization step in AOTO**



**Figure 4.18 Response time reduction vs. optimization step in AOTO**

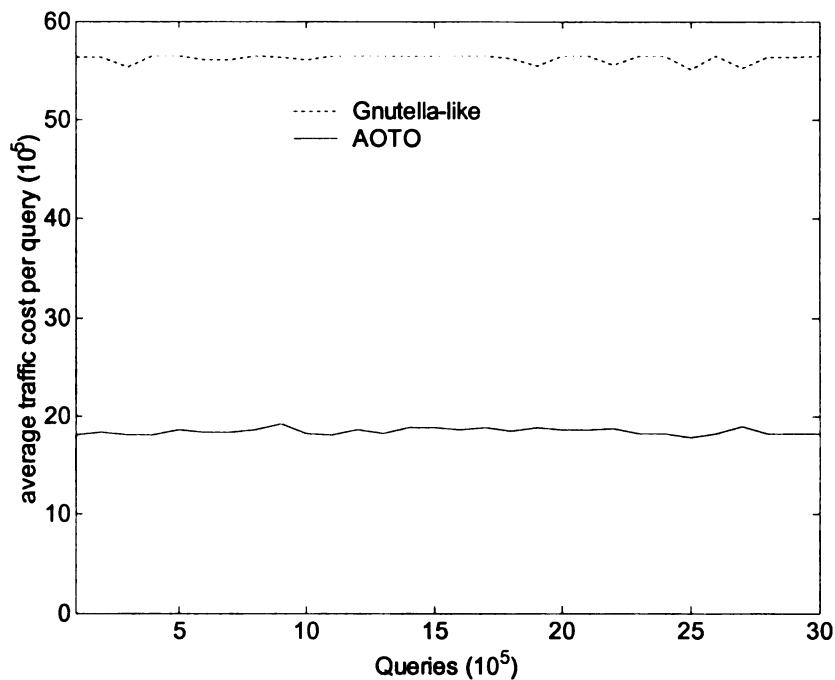
#### 4.2.5 Convergent Speed of AOTO

Figures 4.17 and 4.18 show the traffic cost and response time reduction of AOTO respectively. This simulation is done based on a 27,000 nodes physical topology and 8,000 nodes logical topology. In these figures, the curve of ' $c_n$ -neigh' shows the average traffic cost caused by a query to cover the search scope in x-axis, where in the system the average number of logical neighbors is  $c_n$ . We can see that the both the traffic cost and response time of AOTO decrease when the algorithm is conducted multiple times, where the search scope is all 8000 peers. They both reach a threshold after several steps of optimization. AOTO may reduce traffic cost by around 65% and shorten the query response time by about 35% after 10 steps of optimization.

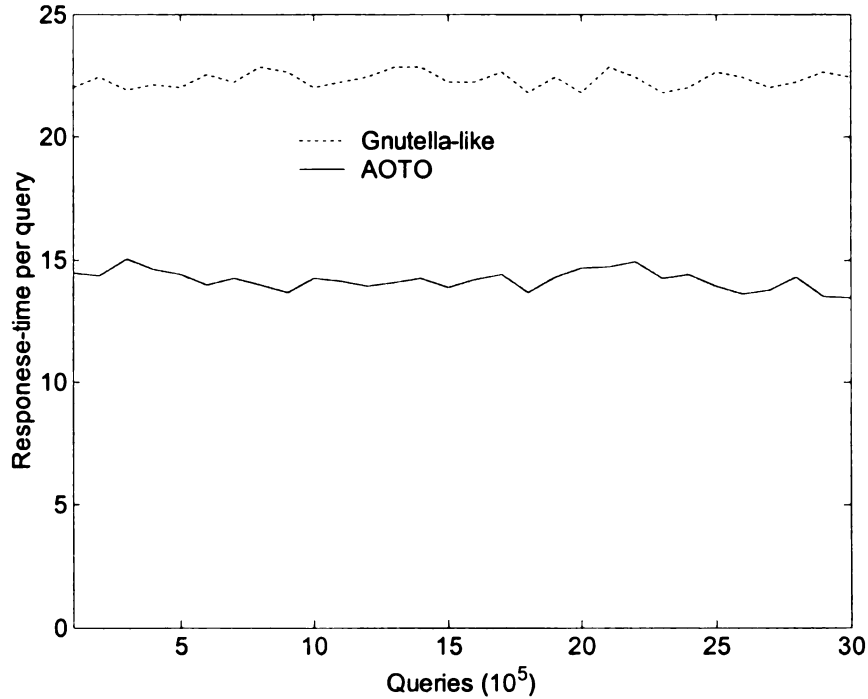
The tradeoff between query traffic cost and response time has been discussed in [97]. P2P systems with a large number of average connections offer a faster search speed while increasing traffic. One of the strengths of AOTO scheme is that it reduces both query traffic cost and response time without decreasing the query success rate.

#### 4.2.6 AOTO in Dynamic Environments

We further evaluate the effectiveness of AOTO in dynamic P2P systems. In this simulation, we assume that peer average lifetime in a P2P system is 10 minutes; 0.3 queries are issued by each peer per minute; and the frequency for AOTO at every peer to conduct optimization operations is twice per minute.



**Figure 4.19 Traffic reduction of AOTO in dynamic P2P environment**



**Figure 4.20 Average response time reduction of AOTO in dynamic P2P environment**

Figure 4.19 shows the average traffic cost per query of Gnutella-like P2P systems and AOTO enabled Gnutella. Note that here the traffic cost includes the overhead needed by each operation in the optimization steps. We can see that AOTO could significantly reduce the traffic cost while retaining the same search scope. In order to keep the same search scope, AOTO may need a larger initial value of TTL. Figure 4.20 shows that with reduction of the traffic, the queries' average response times of AOTO are reduced in a dynamic environment as well.

### 4.3 Location-aware Topology Matching (LTM)

In this section, we introduce our second scheme to address topology mismatch problem: location-aware topology matching [50] (LTM). In LTM, each peer issues a detector in a small region so that the peers receiving the detector can record relative delay information. Based on the delay information, a receiver can detect and cut most of the inefficient and redundant logical links, and add closer nodes as its direct neighbors. Our simulation studies show that the total traffic and response time of the queries can be significantly reduced by LTM without shrinking the search scope. We also show that the overhead of issuing detectors is trivial compared with the query cost savings. LTM consists of three main operations: *TTL2 detector flooding*, *low productive connection cutting*, and *source peer probing*.

#### 4.3.1 TTL2-detector flooding

Based on Gnutella 0.6 P2P protocol, we design a new message type called TTL2-detector. In addition to the Gnutella's unified 23-byte header for all message types, a TTL2-detector message has a message body in two formats as shown in Table 1. The short format is used in the source peer, which contains the source peer's IP address and the timestamp to flood the detector. The long format is used in a one-hop peer that is a direct neighbor of the source peer, which includes four fields: Source IP Address, Source Timestamp, TTL1 IP Address, TTL1 Timestamp. The first two fields contain the source IP address and the source timestamp obtained from the source peer. The last two fields are the IP address of the source peer's direct neighbor who forwards the detector, and the timestamp to forward it. In the message header, the initial TTL value is 2. The payload type of the detector can be defined as 0x82.

**Table 1: TTL2-detector message body**

	Source IP Address				Source Timestamp			
Byte offset	0	3	4	7	8	11	12	15

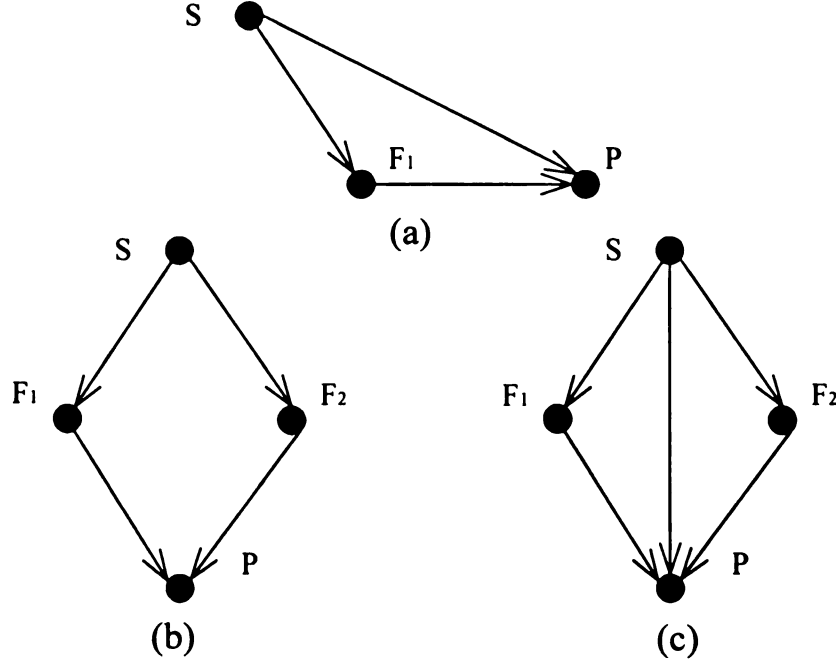
	Source IP Address		Source Timestamp		TTL1 IP Address		TTL1 Timestamp	
Byte offset	0	3	4	7	8	11	12	15

Each peer floods a TTL2-detector periodically. We use  $d(i, S, v)$  to denote the TTL2-detector which has the message ID of  $i$  with TTL value of  $v$  and is initiated by  $S$ . We use  $N(S)$  to denote the set of direct logical neighbors of  $S$ , and use  $N^2(S)$  to denote the set of peers being two hops away from  $S$ . A TTL2-detector can only reach peers in  $N(S)$  and  $N^2(S)$ . We use network delay between two nodes as a metric for measuring the cost between nodes. The clocks in all peers can be synchronized by current techniques in an acceptable accuracy<sup>1</sup>. By using the TTL2-detector message, a peer can compute the cost of the paths to a source peer. As an example in Figure 4.21(a), when peer  $P$  receives a  $d(i, S, 1)$ , it can calculate the cost of link  $SP$  from Source Timestamp and the time  $P$  receives the  $d(i, S, 1)$  from  $S$ . When  $P$  receives a  $d(i, S, 0)$ , it can calculate the cost of link  $SF_1$  from TTL1 Timestamp and Source Timestamp, and  $F_1P$  from TTL1 Timestamp and the time  $P$  receives the  $d(i, S, 0)$  from  $F_1$ . As we can see in an inefficient overlay topology, the peers in set  $N^2(S)$  may receive  $d(i, S, v)$  more than once, such as peer  $P$  in Figure 4.21. If a peer receives  $d(i, S, v)$  multiple times, it will conduct the operations in the second step of LTM, low productive connection cutting.

#### 4.3.2 Low productive connection cutting

<sup>1</sup> Current implementation of NTP version 4.1.1 in public domain can reach the synchronization accuracy down to 7.5 milliseconds [14]. Another approach is to use distance to measure the communication cost, such as the number of hops weighted by individual channel bandwidth.

There are three cases for any peer  $P$  who receives  $d(i, S, v)$  multiple times.



**Figure 4.21 Peer  $P$  receives  $d(i, S, v)$  multiple times in LTM**

*Case 1:*  $P$  receives both  $d(i, S, 1)$  and  $d(i, S, 0)$  as shown in Figure 4.21(a). In this case,  $d(i, S, 1)$  comes from path  $SP$ , while  $d(i, S, 0)$  comes from  $SF_1P$ . The costs of  $SP$ ,  $SF_1$ , and  $F_1P$  can be calculated from the timestamps recorded in  $d(i, S, 0)$  and  $d(i, S, 1)$ . If  $SP$  or  $F_1P$  has the largest cost among the three connections,  $P$  will cut the respective connection. If  $SF_1$  has the largest cost,  $P$  will do nothing. Note that LTM is fully distributed and all peers do the same LTM operations. In the case of  $SF_1$  having the largest cost,  $F_1$  will disconnect this connection.

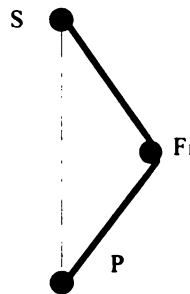
*Case 2:*  $P$  receives multiple  $d(i, S, 0)$ s from different paths as shown in Figure 4.21(b). In LTM,  $P$  randomly takes two of the paths, such as  $SF_1P$  and  $SF_2P$  in Figure 4.21(b), to process at each time. Other paths, if any, will be handled in the next round of optimization. Thus, one important factor to affect the performance of LTM is the frequency for

each peer to issue TTL2-detector messages. We will investigate the optimal LTM frequency, and we expect it is determined by the average peer lifetime and query frequency. Peer P can calculate the costs of  $SF_1$ ,  $SF_2$ ,  $F_1P$  and  $F_2P$ . If  $PF_1$  or  $PF_2$  has the largest cost, P will disconnect it. If  $SF_1$  or  $SF_2$  has the largest cost, P will do nothing. As we have discussed above,  $SF_1$  or  $SF_2$  having the largest cost will be cut by one of the other three nodes.

*Case 3:* P receives one  $d(i, S, 1)$  and multiple  $d(i, S, 0)$ s as shown in Figure 4.21(c). In this case, P will process the path receiving  $d(i, S, 1)$  and one path randomly selected from the multiple paths of  $d(i, S, 0)$ s forming a scenario of *Case 1*.

#### 4.3.3 Source peer probing

For a peer P who receives only one  $d(i, S, 0)$  during a certain time period (e.g., 10 seconds), and  $P \in (N^2(S) - N(S))$ , it will try to obtain the cost of PS by checking its cut list first. If S is not in the list, P will probe the distance to S (see Figure 4.22). After obtaining the cost of PS, P will compare this cost with the costs of  $SF_1$  and  $PF_1$ . If PS has the largest cost, P will not keep this connection. Otherwise, this connection will be created. In the Internet, the cost of SP and the cost of PS may not be the same. We use the cost of PS to estimate the cost of SP.



**Figure 4.22 Source peer probing**

#### 4.3.4 Traffic Overhead of LTM

The simplicity of blind flooding makes it very popular in practice. This mechanism relays a query message to all its logical neighbors, except the incoming peer. For each query, each peer records the neighbors that relay the query to it. Therefore, in the worst case, the same query message can be sent on each link at most twice. For an overlay network with  $n$  peers, we use  $c_n$  to denote the average number of neighbors, and use  $c_e$  to denote the average cost of the logical links. The total traffic caused by a query is less than or equal to  $n c_n c_e$ . In a typical P2P system, the value of  $n$  (more than millions) is much greater than  $c_n$  (less than tens) [78]. So we can view both  $c_n$  and  $c_e$  as constant numbers. Thus, in the flooding-based search, the traffic incurred by one query from an arbitrary peer in a P2P network is  $O(n)$ . As observed in [82], each peer issues 0.3 queries per minute in average. Thus, the per minute traffic incurred by a P2P network with  $n$  peers is  $O(n^2)$ .

Recall that each  $d(i, S, v)$  has a TTL value of 2 in a source peer. So the traffic for one time LTM optimization in all peers is at most  $2n c_n^2 c_e$ . If each peer conducts LTM  $k$  times per minute, the total traffic is  $2kn c_n^2 c_e$ . We find the best value for  $k$  is 2 or 3. Thus, the per minute traffic overhead incurred by LTM to the P2P network is  $O(n)$ . Compared with the query traffic savings, the traffic overhead from LTM is trivial, which will be quantitatively shown later.

One question is why we don't use TTL $_j$ -detector with a TTL of  $j > 2$  in a source peer so that cycles with more than 4 links can be detected and broken. There are two reasons for not doing so. First, if  $j > 2$ , the traffic caused by detector flooding will be increased significantly. Second, if the most expensive connection in a cycle is cut and its cost is not sub-

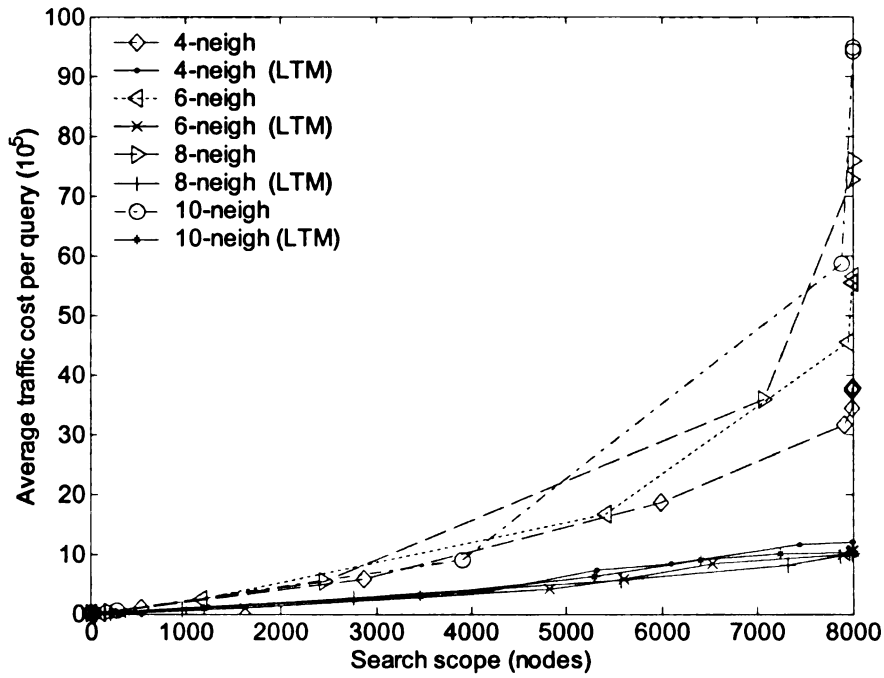
stantially larger than the costs of other links in the cycle, a query initiated from any of the two end peers in the broken cycle will need to traverse a path much more expensive than the cost on the cut connection to reach another end peer.

#### 4.3.5 Effectiveness of LTM in Static Environment

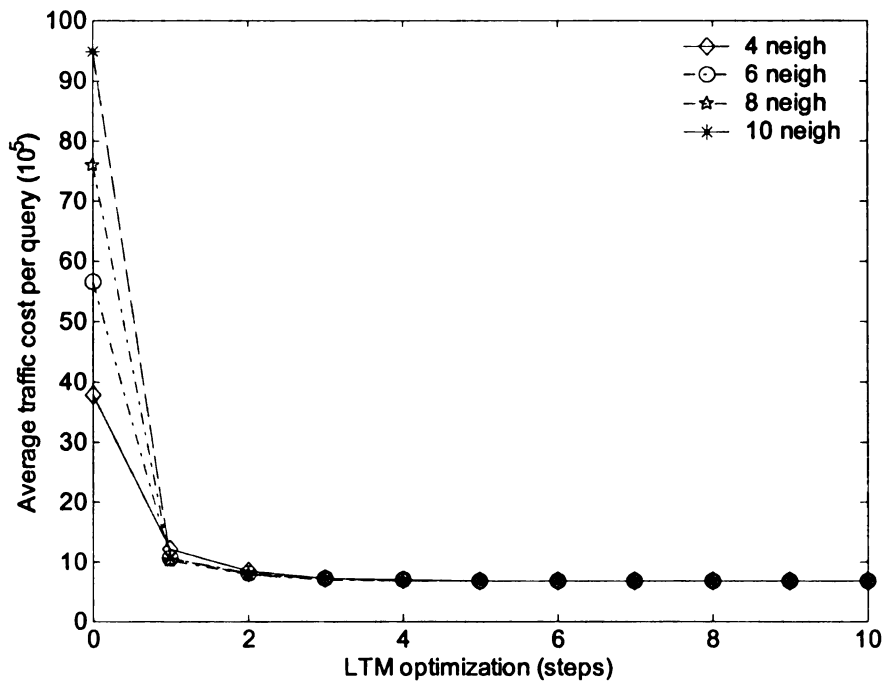
In our first simulation, we study the effectiveness of LTM in a static P2P environment where the peers do not join and leave frequently. This will show that without changing the overlay topology, how many LTM optimization steps are required to reach a better topology matching.

The first goal of LTM scheme is to reduce traffic cost as much as possible while retaining the same search scope. Figure 4.23 compares the traffic cost incurred by the original Gnutella-like system and by the system after one-step LTM optimization. One-step means every peer makes LTM optimization only once. Since this simulation is based on a static P2P environment, we do not include traffic cost incurred by LTM operations. In Figure 4.23, the curve of ' $c_n$ -neigh' shows the average traffic cost caused by a query to cover the search scope in x-axis, where in the system the average number of logical neighbors is  $c_n$ . The dashed curves represent performance results without using LTM, while solid curves represent the results with LTM optimizations. Figure 4.23 shows that to cover the same search scope, one-step LTM reduces the traffic cost significantly, and the reduction rate increases as the search scope increases.

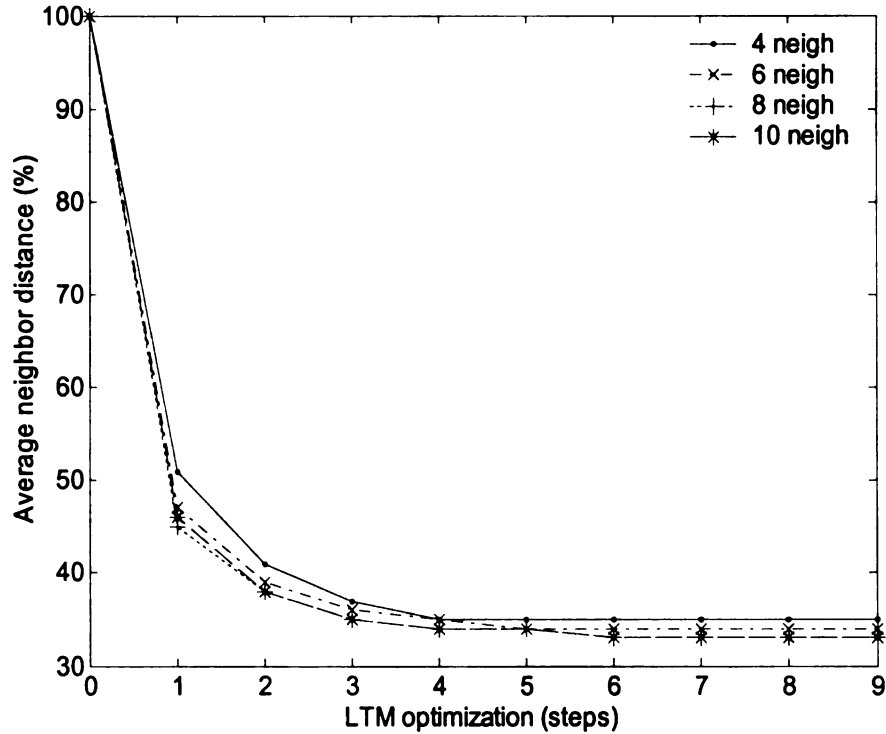
In other words, with a given traffic cost, LTM will increase its search scope. Figure 4.24 shows that the traffic cost decreases when LTM is conducted multiple times, where the search scope is all 8000 peers.



**Figure 4.23 Traffic cost vs. search scope**



**Figure 4.24 Traffic reduction vs. optimization step**



**Figure 4.25 Average neighbor distance vs. optimization step**

We can see that the traffic cost reduction reaches to a threshold after the second or third step LTM optimization. LTM can be convergent as fast as in 2-3 steps.

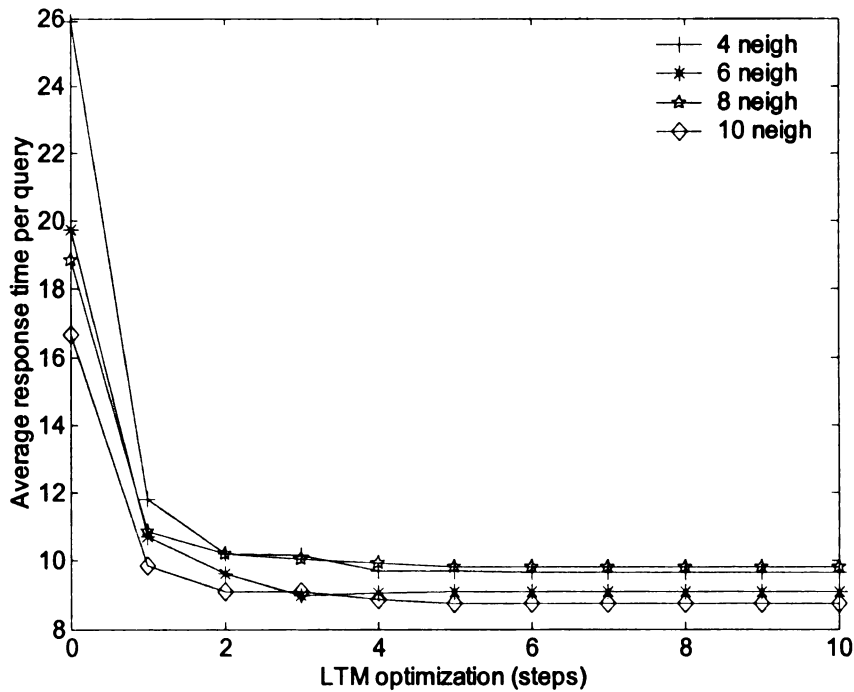
Average neighbor distance reflects effectiveness of LTM on topology match problem. Figure 4.25 shows the average neighbor distance versus LTM optimization steps. Compared with the original Gnutella-like network without LTM scheme (0 optimization steps), one-step LTM optimization reduces AD by about 55%, and more steps of LTM may cut AD to around 65%.

The simulation results in Figure 4.26 and Figure 4.27 show that LTM can effectively shorten the query response time and search latency by about 62% and 55% respectively. LTM scheme reduces both query traffic cost and response time without decreasing the query success rate.

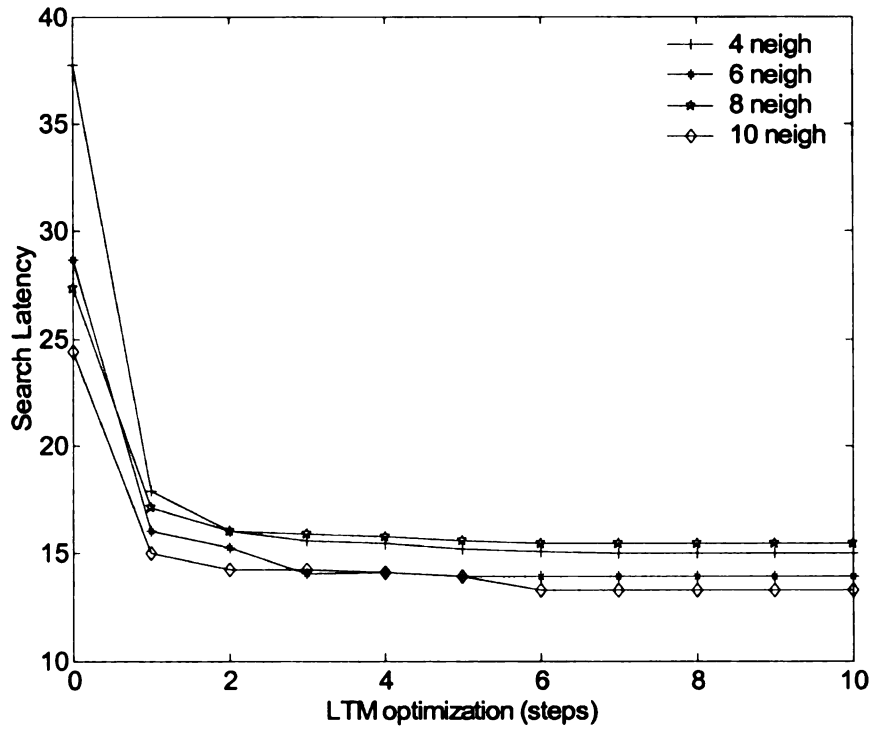
Our other simulation results, which are not presented due to the page limitation, also show that different densities of logical peers or physical nodes will not impact the effectiveness of LTM. The average traffic cost is only proportional to the average number of neighbors and average cost logical links, which is consistent with previous analysis.

#### 4.3.6 LTM in Dynamic Environment

We further evaluate the effectiveness of LTM in dynamic P2P systems and explore the best frequency for each peer to conduct LTM. We first discuss the performance impact of the will-cut list and the cut list. The average number of logical neighbors we use is 6.



**Figure 4.26 Average response time vs. optimization step**

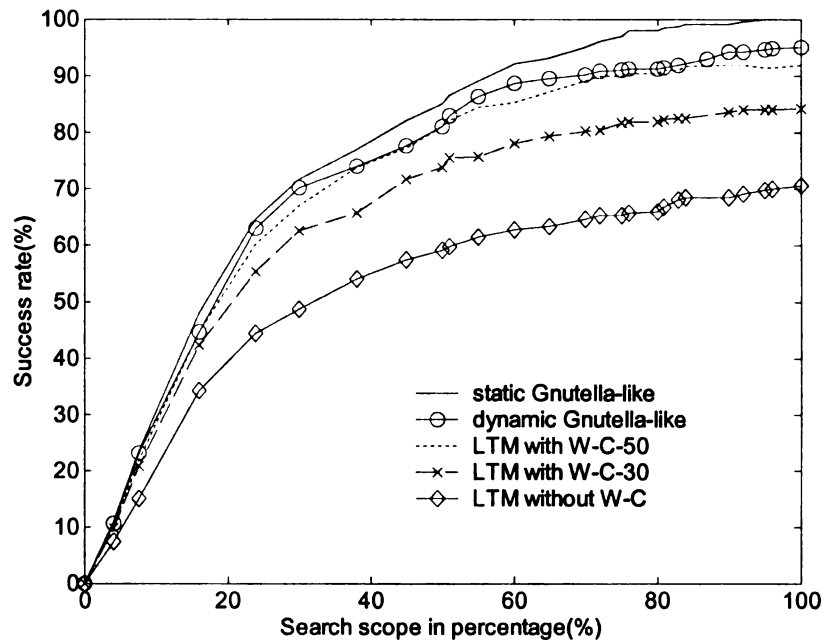


**Figure 4.27 Average search latency vs. optimization step**

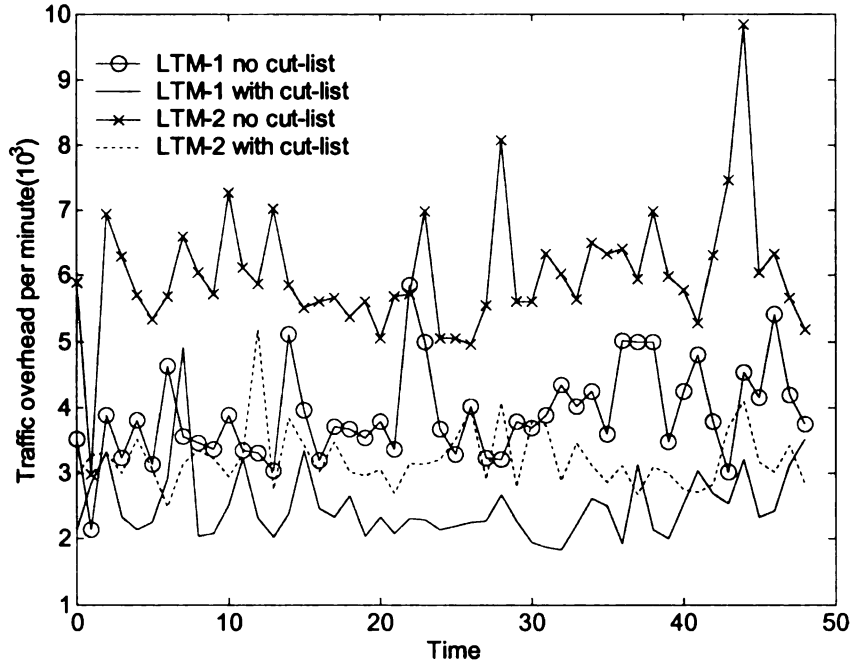
From our simulation results in dynamic environments, we found that with the same search scope the query success rate in dynamic environments is decreased by about 5% compared with the static environment, as shown in Figure 4.28 (compare curves of static Gnutella-like and dynamic Gnutella-like). One extreme case is when the search scope is 100%, which means that each query can reach all peers and we guarantee the query result is available in at least one of the peers. The search success rate is expected to be 100% in this case, but it is only 95%. The reason of the 5% loss in query success rate is that the query responses cannot be returned due to peers' dynamic leaving behavior. We call this phenomena response loss problem.

If we don't use the will-cut list in LTM, a connection will be cut immediately when it is found to be a slow connection, which will cause a very serious response loss problem

because many responses may not be returned due to the cut connections. The curve of LTM without W-C in Figure 4.28 shows that the query success rate is significantly decreased by 30-40% without using the will-cut list. The LTM is conducted once every minute in this simulation. Retaining query success rate is the reason we design the will-cut list, each of which can hold 20 connections in our simulation. The up to 20 slow connections will not be used to forward queries, but only used to return query results. The lifetime of the connections in a will-cut list determines the query success rate. In Figure 4.28, a curve of LTM with W-C-n means the lifetime of a will-cut connection is n seconds. We can see that the query success rate can be retained if the connections can be kept in the will-cut list for 50 seconds.



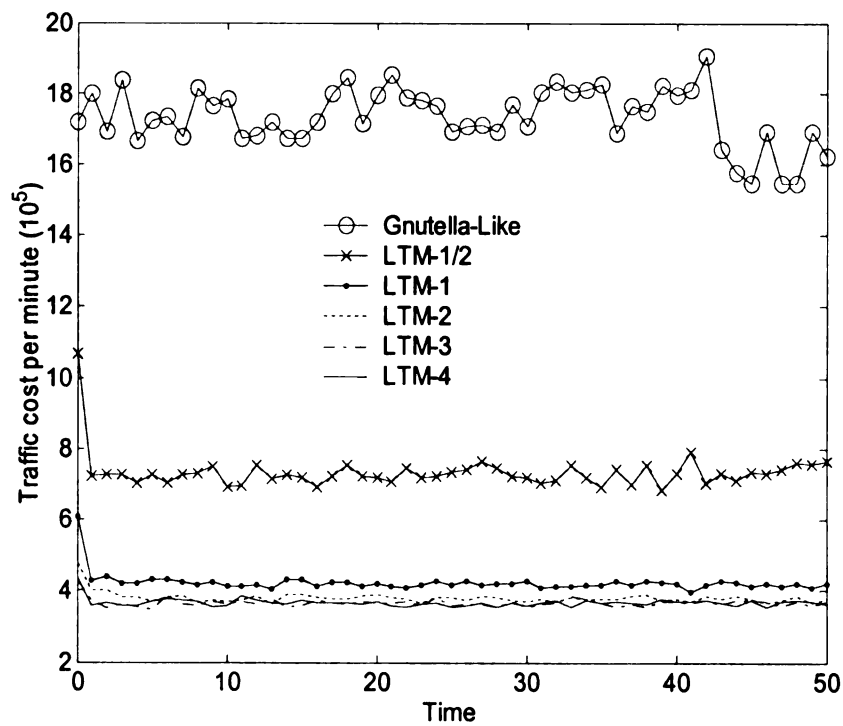
**Figure 4.28 Effectiveness of will-cut list**



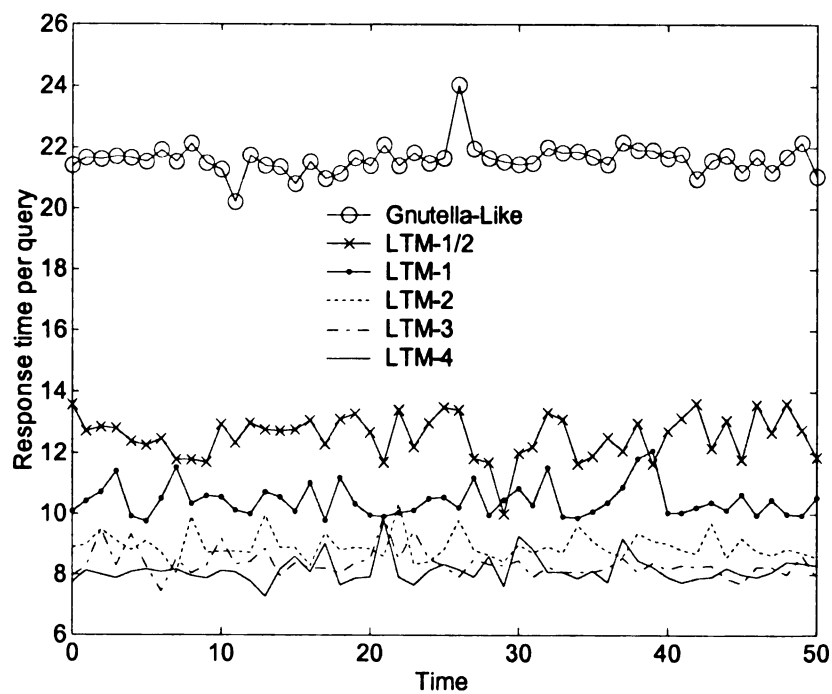
**Figure 4.29 Effectiveness of cut list**

Another thing which deserves some words is the design of cut list. If we don't use the cut list, a connection that has just been cut may be established again. Thus the LTM optimization rate will be limited. Figure 4.29 compares the overhead incurred by LTM with and without the use of the cut list. The fluctuations of the curves represent the dynamic nature of the network as time goes. The curve of LTM-k means each peer conducts LTM for k times per minute. We can see that the use of the cut list reduces traffic overhead by about 50% compared with the case without using the cut list.

We use the will-cut list and the cut list in this part of simulation. Compared with a Gnutella-like system, Figure 4.30 and Figure 4.31 show the effectiveness of LTM on reducing average traffic cost and query response time.



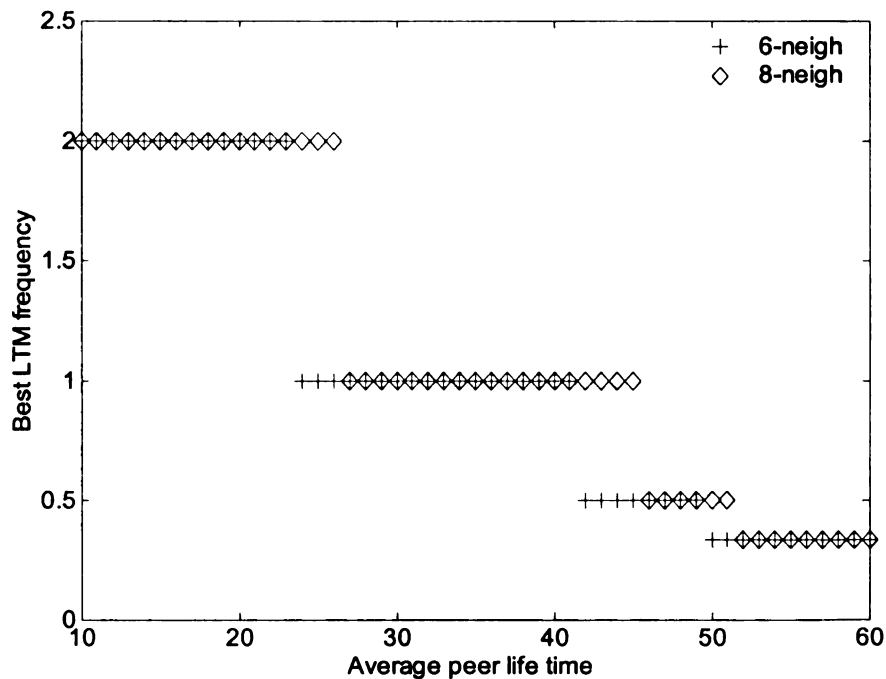
**Figure 4.30 Total traffic vs. LTM frequency**



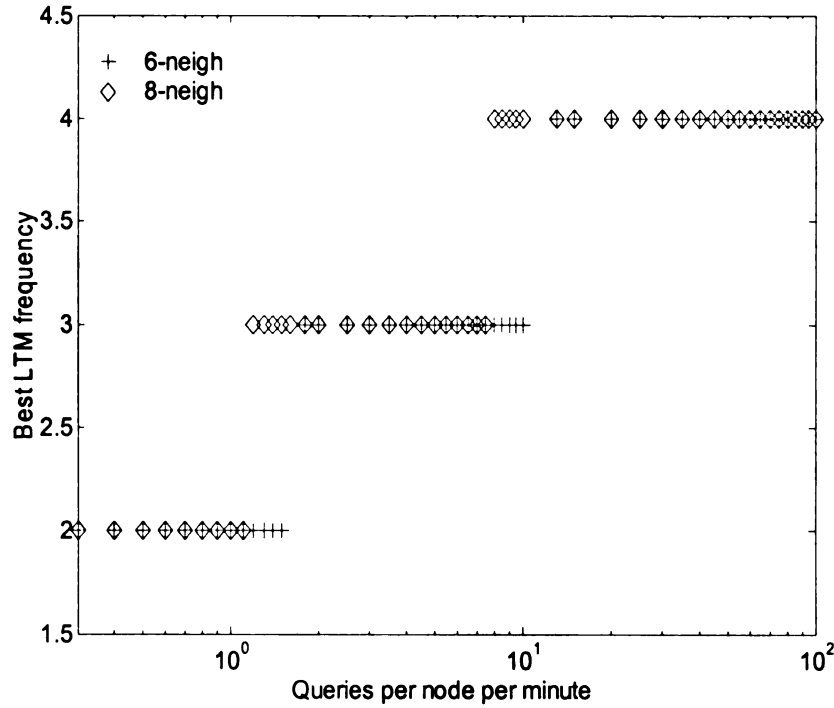
**Figure 4.31 Response time vs. LTM frequency**

Since LTM adds some traffic overhead due to the TTL2 detector flooding, there exists an optimal frequency for each peer to conduct LTM independently. We simulate LTM in different frequencies ranging from 1/4 to 4 times every minute. We consider a frequency to be optimal if the next higher frequency does not increase the optimization by more than 3% compared with the current frequency. Results in Figure 4.30 and Figure 4.31 show that under the assumption that peer average lifetime in a P2P system is 10 minutes, and 0.3 queries are issued by each peer per minute, the optimal frequency for every peer to conduct LTM is twice per minute. With this frequency, about 75% reduction on traffic cost and 65% reduction on response time can be achieved.

As we have mentioned, different values of peer average lifetime and query frequency have been presented by previous studies [18, 72, 78, 82]. We further tune the two parameters (average lifetime and query frequency) in our simulation.



**Figure 4.32 Optimal LTM frequency vs. average peer lifetime**



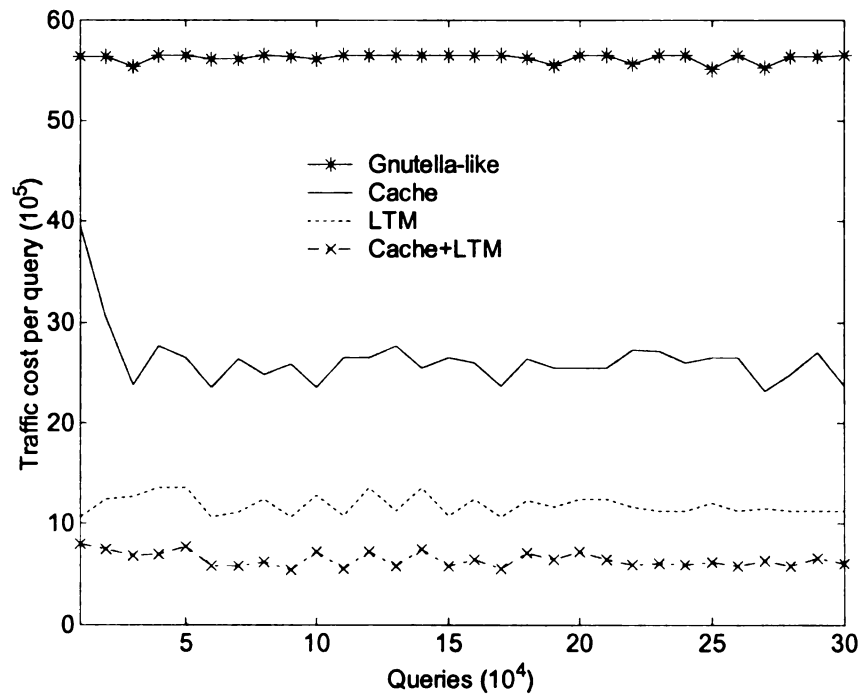
**Figure 4.33 Optimal LTM frequency vs. average query frequency**

Figure 4.32 shows that LTM can be conducted less frequently if peer average life-time is longer. Figure 4.33 shows that LTM should be conducted more frequently if more queries are issued. Both figures show that a larger average number of neighbors require a higher LTM frequency.

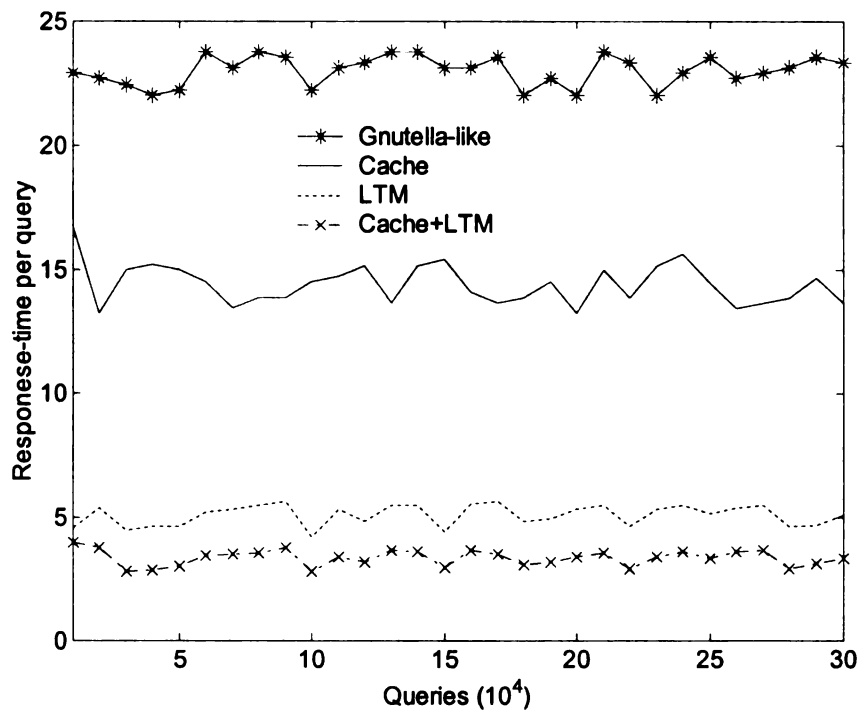
#### 4.3.7 Combining LTM and Query Index Caching

We compare the traffic cost and response time in a Gnutella-like system without any optimization, with query index caching only, with one-step LTM optimization only, and with one-step LTM optimization plus query index caching.

Results in Figure 4.34 and Figure 4.35 show that by combining LTM and query index caching the traffic cost is reduced by about 10 times without shrinking the search scope, and the average query response time is reduced by about 7 times.



**Figure 4.34 Traffic cost of four schemes**



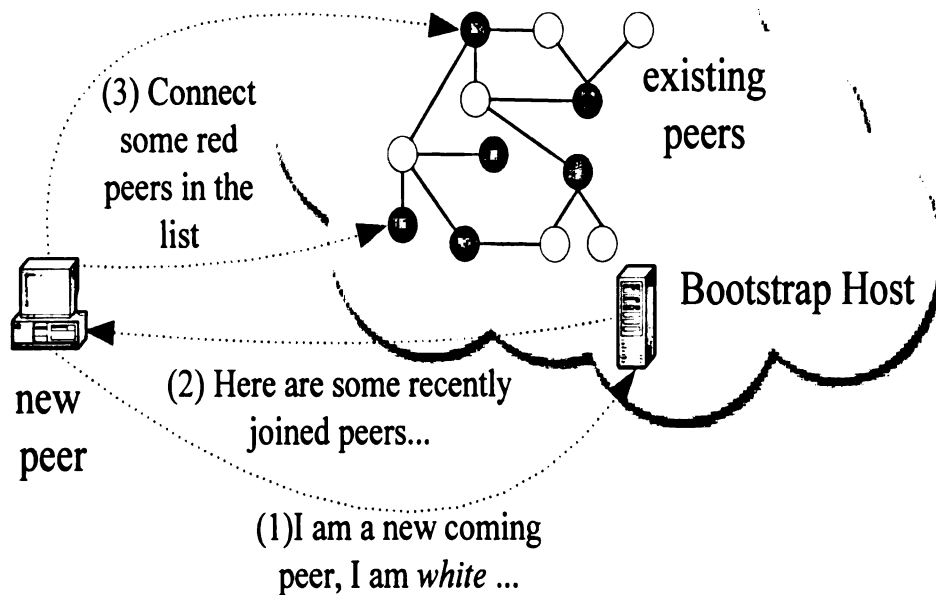
**Figure 4.35 Average response time of four schemes**

## **4.4 Scalable Bipartite Overlay (SBO)**

We have presented AOTO and LTM. Since AOTO only work with peers' one hop away neighbor, its convergent speed is relative slow. Thus, its effectiveness is degraded in highly dynamic systems. LTM has a faster speed compared with AOTO, but it needs to synchronize all the peering nodes. In this chapter, we introduce SBO, which employs an efficient strategy for distributing optimization tasks in peers with different colors. In SBO, each joining peer is assigned a color so that all peers are divided into two groups of white or red colors, respectively. White peers probe neighbor distances and reports the information to the red neighbors and red peers compute efficient forwarding paths. A white peer that is not on forwarding paths of a red peer tries to find a more efficient red peer to replace this neighbor. The topology construction and optimization of SBO consist of four phases: bootstrapping a new peer, neighbor distance probing and reporting, forwarding connections computing, and direct neighbor replacement.

### **4.4.1 Design of SBO**

In the first phase of SBO, each joining peer is randomly assigned a color so that all peers are divided into two groups with white or red colors, respectively. Each peer is only connected with peers in a different color. In the second phase, each white peer probes its distances with all its red neighbors and reports the information to the red neighbors. In the third phase, each red peer computes efficient forwarding paths so that the same search scope can be retained without the need to flood a query to all neighbors. In the fourth phase, a white peer who is not on the forwarding path tries to find a more efficient red peer to replace its current neighbor.



**Figure 4.36 Bootstrapping a new peer**

#### **Phase 1: bootstrapping a new peer.**

When a new peer is joining the P2P system, it will randomly take an initial color: red or white. A peer should keep its color until it leaves, and again randomly select a color when it rejoins the system. Thus, each peer has a color associated with it, and all peers are separated into two groups, red and white. In SBO, a bootstrap host will provide the joining peer a list of active peers with color information. The new joining peer then tries to create connections to the different color peers in the list. Figure 4.36 illustrates a new peer's joining process. In such a way, all the peers form a bipartite overlay, in which a red peer will only have white peers as its direct neighbors, and vice versa.

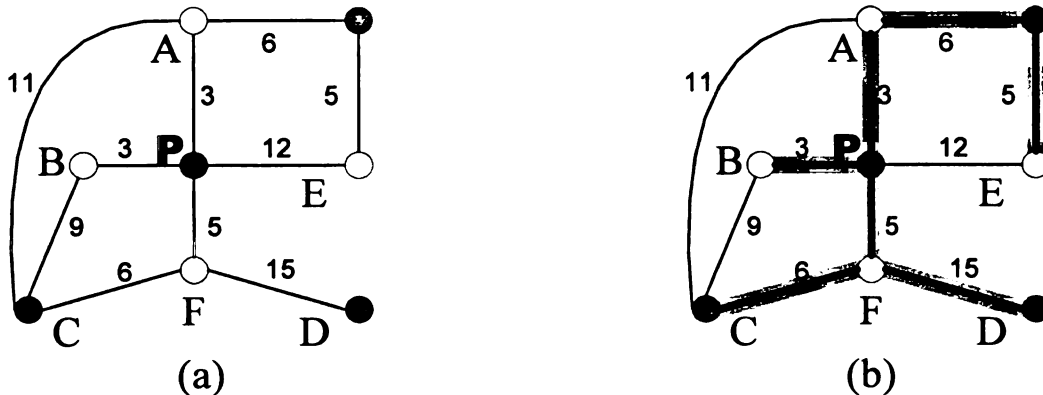
Once a peer has joined the P2P system, it will periodically ping the network connections and obtain the IP addresses of other peers in the network, which will be used to make new connections for the peer's rejoining or in the case that the peer loses some of

the connections with its neighbors due to the neighbors' departure or failure, or the faults in the underlying networks

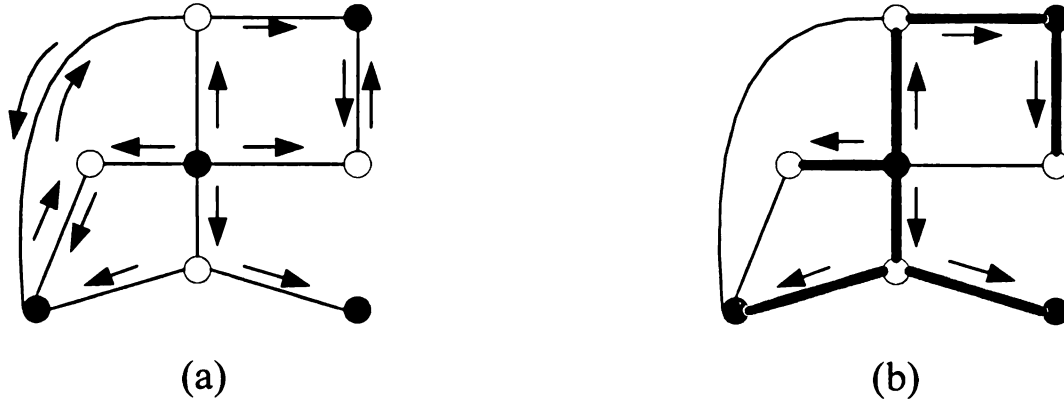
**.Phase 2: neighbor distance probing and reporting by white peers**

We modify the Limewire implementation of Gnutella 0.6 P2P protocol by adding one routing message type for a peer to probe the cost with its neighbors. Each white peer probes the costs with its immediate logical neighbors and forms a neighbor cost table, and sends this table to all its neighbors who are all red peers. The impact of the frequency of the white peers' probing and cost table reporting operation will be discussed in more detail later.

Since each red peer,  $P$ , receives the cost table from its white neighbors about its all red neighbors, the red peer  $P$  has the information to obtain the overlay topology including  $P$  itself,  $N(P)$ , and  $N^2(P)$ , as illustrated in Figure 4.37 (a). Note that in SBO the overlay forms a bipartite topology, so there is no connections between any pairs of peers in  $N^2(P)$ . Thus, we only require all the white peers to probe the costs to their neighbors and send out the cost tables. There is no need for the red peers to probe the distance.



**Figure 4.37 A red peer  $P$  has topology of  $(P + N(P) + N^2(P))$ , and computes the MST**



**Figure 4.38 A red peer computes the efficient forwarding paths**

### **Phase 3: forwarding connections computing by red peers**

Based on obtained neighbor cost tables, a minimum spanning tree (MST) can be built by each red peer, such as P in Figure 4.37 (b). Since a red peer builds a MST in a two-hop diameter, a white peer does not need to build a MST. The thick lines in the MST are selected as forwarding connections (FC), while the rest lines are non-forwarding connections (NFC). Queries are only forwarded along FCs. For example, in Figure 4.37(b), P will send/forward queries to A, B and F, but not E. Peer P also informs E that E is a non-forwarding neighbor. This information will be used by E in Phase 4, i.e., direct neighbor replacement.

Figure 4.38(a) illustrates how the query message from P is flooded along the connections based on Figure 4.37(a). We can see many message duplications, i.e. RK problem. The total traffic cost incurred by the query is:  $3+6+5+5+12+3+5+6+9+9+15+11+11=100$ . After FC computing in Figure 4.37(b), the traffic cost incurred by this query becomes:  $3+6+5+3+5+6+15=43$  as shown in Figure 4.38(d).

Although FC computing can reduce a lot of traffic while retaining the same search scope, as we described earlier, the price is to scarify query response time, or the query

latency. For instance, P issues a query, and E has the desired data. The response time in Figure 4.38(a) is  $2 \times 12 = 24$ . After FC computing, the response time becomes  $2 \times (3 + 6 + 5) = 28$ . Based on this observation, we will further improve our FC selecting algorithm later in this section.

#### **Phase 4: direct neighbor replacement by white peers**

This operation is only conducted by white peers. The goal of neighbor replacement is to alleviate the topology mismatching problem, or RN problems. As we have explained, solving RN problem is essential since it will not only reduce message duplications and traffic cost, but also shorten the response times.

After computing a MST among the peers within two hops, a red peer P is able to send its queries to all the peers within two hops. Some white peers become non-forwarding neighbors, such as E in Figure 4.37. In this case, for peer E, P is no longer its neighbor. In the phase of direct neighbor replacement, a non-forwarding neighbor, E, will try to find another red peer being two hops away from P to replace P as its new neighbor.

Peer P will send the neighbor cost tables it collected from A, B and F to the non-forwarding neighbor E so that E has enough information to find another neighbor to form a more efficient topology. Having received the cost tables, E can obtain the overlay topology among P and the peers  $N(P)$  and  $N^2(P)$ . In the design of SBO, E will probe the round trip times (RTTs) to all the red peers in  $N^2(P)$  and sort the red peers according to their RTTs. Peer E then selects the one with the smallest RTT, e.g., peer D in Figure 4.39(a). There are three cases for peer E who finds D as its nearest red peer.

*Case 1:* The delay of ED is smaller than that of EP. The connection of ED will be created, and D becomes E's direct logical neighbor. The connection EP will be put into E's

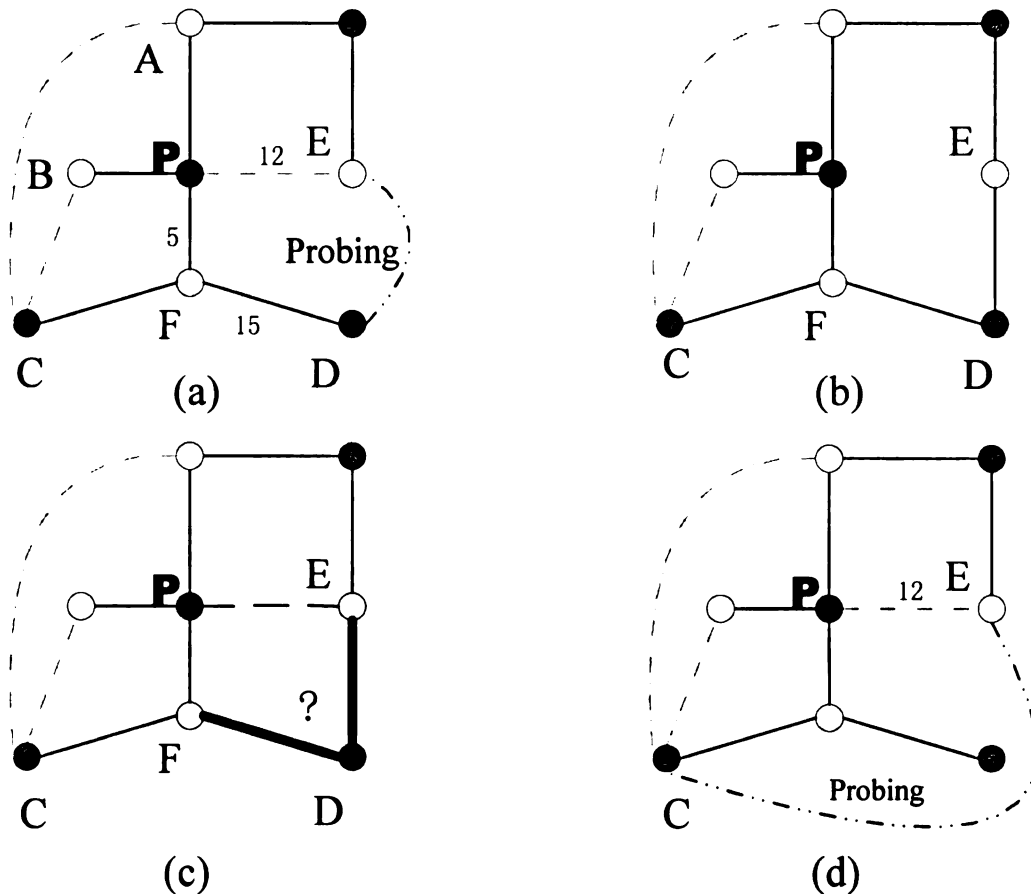
will-cut list that is a list of connections to be cut later. A connection in a will-cut list will be disconnected when it has been in the list for a certain period of time. A peer will not send or forward any queries to the connections in its will-cut list.

The reason for E not to disconnect EP immediately is that some query responses might be sent back along the overlay path EP for some earlier queries. Disconnecting non-forwarding connections, such as EP, immediately may cause serious response loss problem. Figure 4.39(b) is the topology after E connects with D, and disconnects with P after a timeout period.

*Case 2:* The delay of ED is larger than that of EP, but is smaller than the larger one of PF and FD. For example, if  $ED=13$  in Figure 4.39(c),  $12 < ED < 15$ . In this case, E will create the connection of ED and treat D as its direct neighbor. Peer E will not put connection EP into its will-cut list until it sends its neighbor cost table to D so that D still thinks the connection of EP exists. Note that the algorithm is completely distributed. Thus, when red peer D conducts the FC computing, F will become D's none-forwarding neighbor. The white peer F will conduct the same operations as what peer E has done, and may try to find a better red peer to replace node D as its neighbor.

*Case 3:* If ED has the largest delay among EP, PF and FD, peer E will pick the second nearest peer in  $N^2(P)$ , such as C in Figure 4.39(d), and repeat the above process until it finds a better node to replace P as its neighbor, or until it has tried all the peers in  $N^2(P)$ .

The first three operations are relatively straightforward, so we do not provide the detailed pseudo code, and the pseudo code of direct neighbor replacement operation is as below.



**Figure 4.39 An example of neighbor replacement**

**For a white peer  $i$**

**For each peer  $j$  in white peer  $i$ 's non-forwarding neighbor list**

Replaced = false;

List=all the two hop away red neighbors of  $j$ ,  $N^2(j)$ ;

Peer  $i$  pings all the peers in List;

Add peers' RTT information to List;

**While** List is not empty and Replaced = false

remove the peer  $h$  with smallest RTT from List;

**if**  $RTT_{ih} < RTT_{ij}$  {replace  $j$  by  $h$  in  $i$ 's neighbor list;

```

    Put  $j$  into will-cut list;

    Replaced = true;}

else

    Common_list=all common neighbors of peer  $j$  and  $h$ ;

    While Common_list is not empty and Replace=false

        Randomly remove a peer  $k$  from Common_list;

         $RTT_k = \max\{RTT_{kj}, RTT_{kh}\};$ 

        if  $RTT_{ih} < RTT_k$ 

            {add  $h$  to  $i$ 's neighbor list;

            remove  $j$  from  $i$ 's neighbor list right after

             $i$  finds out  $jk$  or  $kh$  is disconnected;

            Replaced = true;};

        End While;

    End While;

End For;

```

#### 4.4.2 Further Improvements

Previous studies have shown that queries and queried data have significant locality [76, 89]. A small number of peers issue a large portion of the queries and the 5% of the files accounts for 50% of all transfers. Peers' behaviors are different in the query frequency and response frequency. We define a query-heavy peer who issues queries frequently, and a response-heavy peer who often responds queries. We have discussed in the previous section that reducing RK duplication may lead to increase query response time. To avoid disconnecting a path from/to a query/response heavy peer, we further improve SBO by

keeping some *single direction connections* (*SDC*). Below we define *Query-heavy peer* and *Response-heavy peer*.

***Query-heavy peer.*** If the number of queries that a peer have issued or forwarded is 5 times more than the average number of queries in last minute (the number of 5 times is selected based on our simulation), it is defined as a query-heavy peer. In our simulation, with an average number of neighbors being 6, initial TTL=7, average peer lifetime of 10 minutes, and query frequency of 0.3 queries issued per peer per minute, we measured that the average number of queries processed (issued and forwarded) by each peer is about 15 to 25 per second. Thus, a peer is identified as a query-heavy peer if it processed more than 75 queries per second.

***Response-heavy peer.*** In Gnutella protocol v0.6, Query Hit (response) messages are sent along the same path that carried the incoming query message. In our simulation, a peer delivers or forwards 3 responses per minute in average. In SBO, a peer processed more than 20 responses in last minute is defined as a response-heavy peer (The number of 20 responses is selected based on our simulation).

***Single Direction Connections (SDC).*** Every peer in SBO will monitor its own status. If a peer finds itself a query/response-heavy peer, it will report its status to all its neighbors. Thus, when a red peer computes FCs to form the forwarding paths, a white neighbor who is not a forwarding peer may be a query- or response-heavy peer. The connection between the red peer and the white peer will be set as a *SDC*. For example, if peer E in Figure 4.39(b) is a response-heavy peer, instead of setting PE as a non-forwarding connection, it will set PE as a SDC:  $P \rightarrow E$ , where P will send/forward query messages to E while E will not send/forward any query messages to P. In this case, E will still do its

neighbor replacement operation. The SDC:  $P \rightarrow E$  will be disabled when E is no longer a response-heavy peer. If E is a query-heavy peer, connection PE will be set as SDC:  $E \rightarrow P$ , where E will send/forward query messages to P while P will not send/forward any query messages to E.

#### 4.4.3 Traffic Overhead of SBO Optimizations

One optimization step of SBO includes all white peers' neighbor distance probing/reporting and neighbor replacement. In the worst case, each white peer, P, needs to probe every peer in  $N^2(P)$ . It is reasonable to assume that the traffic overhead of peer A probing peer B is equal to a query message traversing the connection AB twice. If each peer conducts SBO optimization operation k times per minute, the total traffic overhead per minute is:

$$k \times \left( \frac{n}{2} \times (2c_n c_e + c_n c_e + 2c_n^2 c_e) \right) = \frac{k c_n c_e (3 + 2c_n)}{2} n$$

Our simulation results will show that the optimal value for k is less than 1, so the per minute traffic overhead incurred by SBO to the P2P network is  $O(n)$ . Compared with the query traffic savings, the traffic overhead from SBO optimization is relatively trivial.

#### 4.4.4 Property analysis of SBO operations

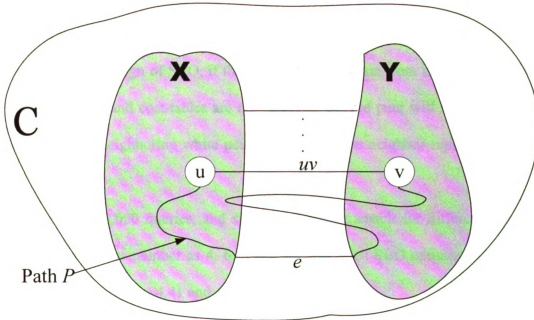
We are going to prove that SBO operations will not increase the number of components of a graph.

**Theorem:** Given a bipartite graph  $G = (V, E)$ , the SBO optimization operations will not increase the G's component number.

**Proof:** We prove by contradiction. Suppose our claim is false. Then, there exists at least one component  $C$ , where  $C$  is a subgraph of  $G$ , which could be disconnected by the SBO operations. Suppose  $C$  is disconnected into two parts,  $X$  and  $Y$ , after SBO operations, as shown in Figure 4.40.

Before the SBO optimization, there must be one or more edges between  $X$  and  $Y$  since  $C$  is connected. Let  $M$  denotes the set of the edges between  $X$  and  $Y$ . Among all these edges in  $M$ , we choose the shortest one,  $uv \in M$ . Here we assume there are no exact equal length edges in the system, so  $uv$  is the only shortest edge in  $M$ .

Since  $G$  is a bipartite graph, peer  $u$  and  $v$  must have different colors. Without loss of generality, we can assume  $u$  is red and  $v$  is white.  $C$  is disconnected after SBO operations means that none of the edges in  $M$ , including  $uv$ , is selected as  $u$ 's forwarding path.



**Figure 4.40 Proof of the property of SBO operations**

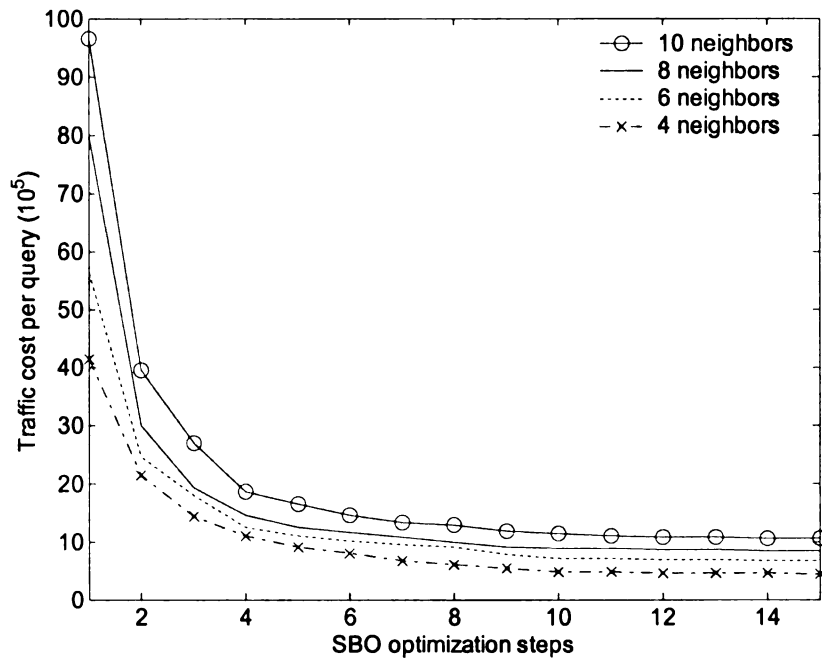
We know that the red peer  $u$  employs a MST algorithm, such as kruskal algorithm, in FC computing operation. Because  $v$  is  $u$ 's one hop neighbor,  $v$  must be included in  $u$ 's MST. In kruskal algorithm, edges are sorted from shortest to longest. The edge  $uv$  is not selected by MST means that there is already another path  $P$  ( $uv \notin P$ ) between  $u$  and  $v$ , and the length of each edge in  $P$  is shorter than  $uv$ . As  $P$  is between  $X$  and  $Y$ , at least one of the edges in  $P$ , say edge  $e$ , belongs to  $M$ . Thus, we have  $e < uv$ , which is a contradiction to our choice that  $uv$  is the shortest edge in  $M$ . ■

#### 4.4.5 Effectiveness of SBO in Static Environments

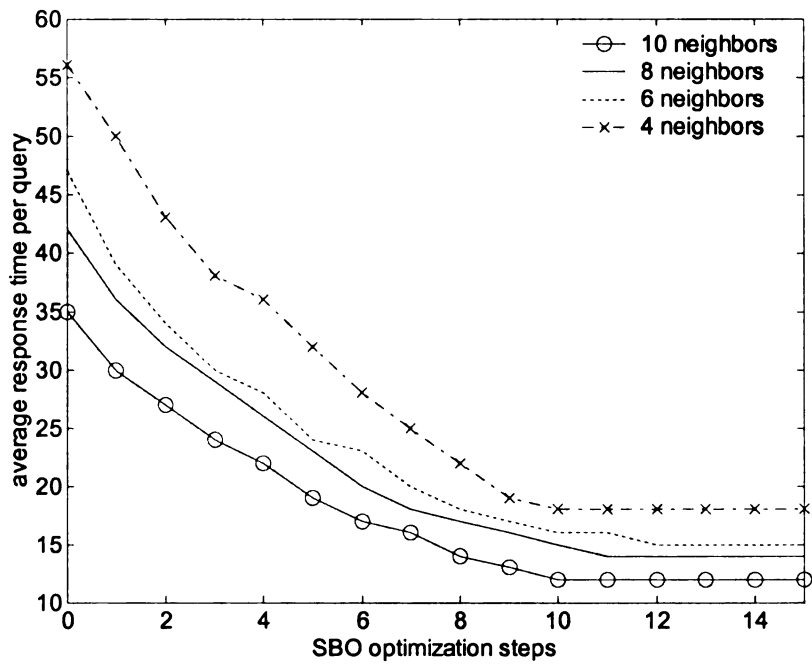
We study the effectiveness of SBO in a static P2P environment firstly. This will show that without changing the overlay topology, how many SBO optimization steps are required to reach a better topology matching. Here one step we mean each red peer collects the neighbor cost tables from its neighboring white peers, and computes the efficient forwarding connections, and its neighbors finish neighbor replacement operations, if needed.

Note that in the design of SBO, if the reported information from all neighbors including neighbor status and cost-tables are not changed, the red peer will not compute FCs. Consequently the neighboring white peers will not do the neighbor replacement operations.

We generate 500,000 queries, and simulate flooding search for different topologies with average neighbor number as 4, 6, 8 and 10 after each SBO optimization step, and show the results in Figures 4.41 and 4.42.



**Figure 4.41 Traffic reduction vs. optimization steps**



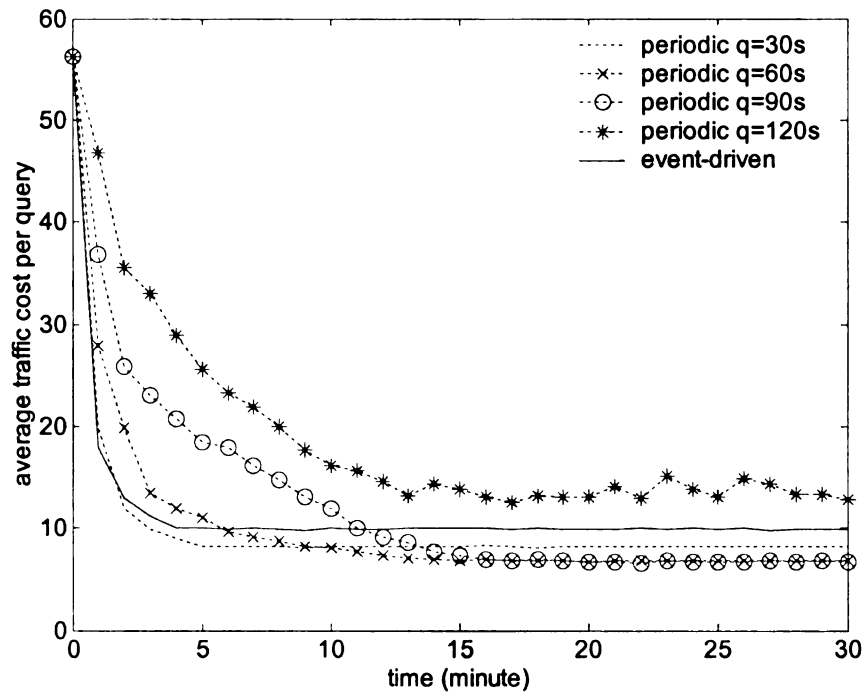
**Figure 4.42 Response time vs. optimization steps**

Figure 4.41 shows that the traffic cost decreases when optimization operations of SBO are conducted multiple times, where the search scope is all 7,000 peers. To cover the same search scope, SBO reduces the traffic cost significantly in first two optimization steps. We can see that the traffic cost reduction reaches to a threshold after eight to ten steps of SBO optimization. The simulation results in Figure 4.42 show that SBO can effectively shorten the query response time by about 60% in first 10 optimization steps. SBO reduces both query traffic cost and response time without decreasing the query success rate.

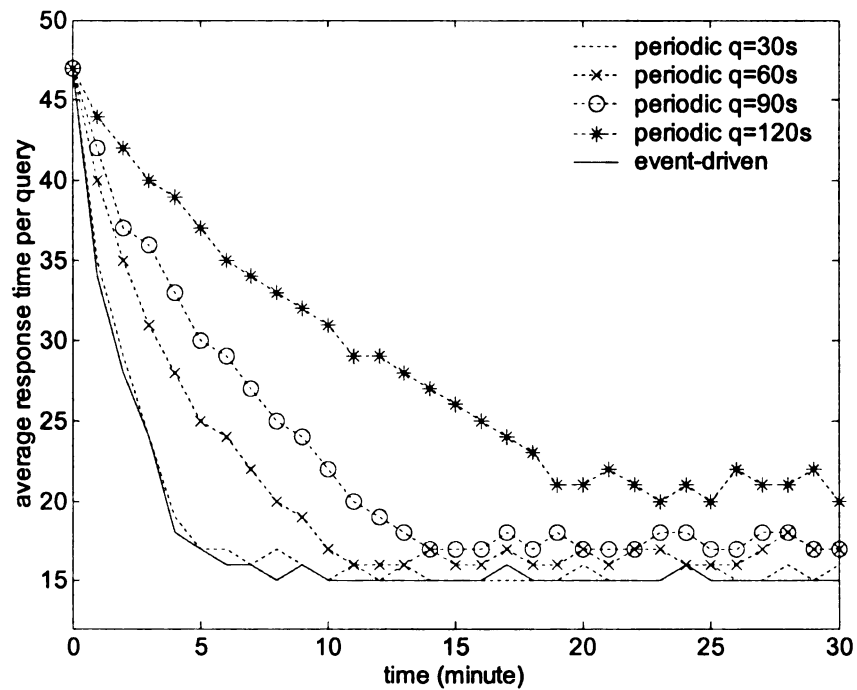
Our other simulation results, which are not presented due to the page limitation, also show that different densities of logical peers or physical nodes will not impact the effectiveness of SBO. The average traffic cost is only proportional to the average number of neighbors and average cost of logical links, which is consistent with previous analysis.

#### **4.4.6 Frequency of SBO Optimizations**

In SBO, there are two ways for a white peer to decide when to conduct neighbor probing and reporting, namely periodic and event-driven. In periodic approach, each white peer conducts neighbor distance probing at every certain period of time,  $q$ . After probing the distances to all the neighbors, a white peer sends the cost table to its neighboring red peers. In event-driven approach, a white peer produces and sends an updated cost table to its neighboring red peers only if there is a change on its logical connections with its neighbors, such as on a neighbor's leaving or on a peer's joining as its new neighbor.



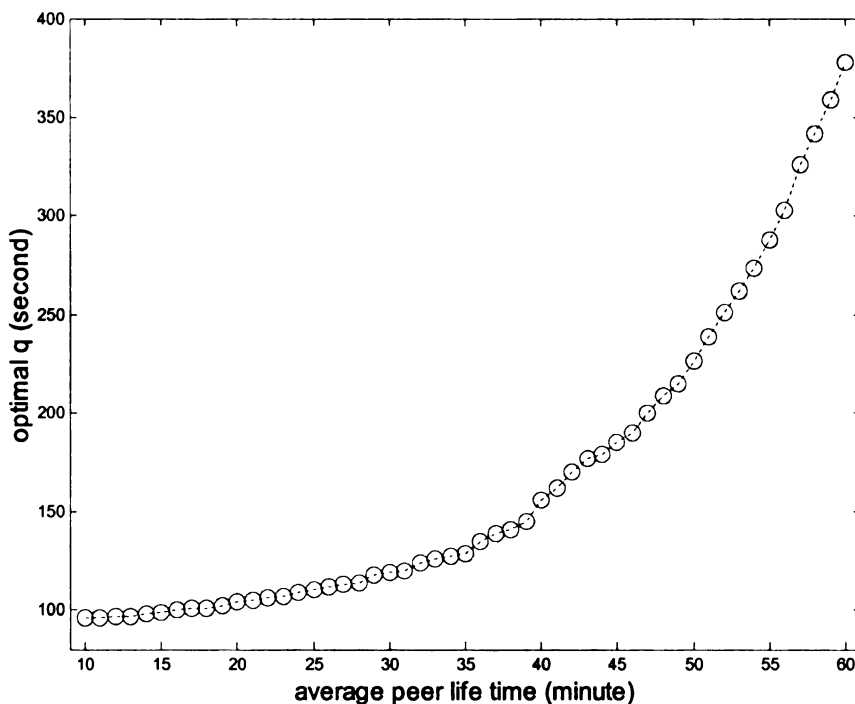
**Figure 4.43 Traffic cost reduction of SBO in dynamic P2P environments**



**Figure 4.44 Response time reduction in dynamic P2P environments**

The value  $q$  is a critical factor for the performance of periodic approach. We have investigated the impact of different values of  $q$  ranging from 20s to 600s. Figures 4.43 and 4.44 show the results on some representative samples of  $q$  at 30s, 60s, 90s, and 120s, respectively, where x-axis indicates the time elapsed since the first probing or event occurred. A small  $q$  leads to a fast convergent speed. However, if  $q$  is too small, e.g.  $q=30$ , peers will conduct the optimization operations too often, making the overhead keep growing when the reduction of the traffic cost and response time have already reached a threshold.

The value of  $q$  should be able to adaptive to the average peer lifetime in order to achieve optimal performance. Figures 4.43 and 4.44 also show that the periodic approach with  $q=90$ s outperforms even-driven approach on traffic reduction.



**Figure 4.45 Optimal SBO optimization time interval**

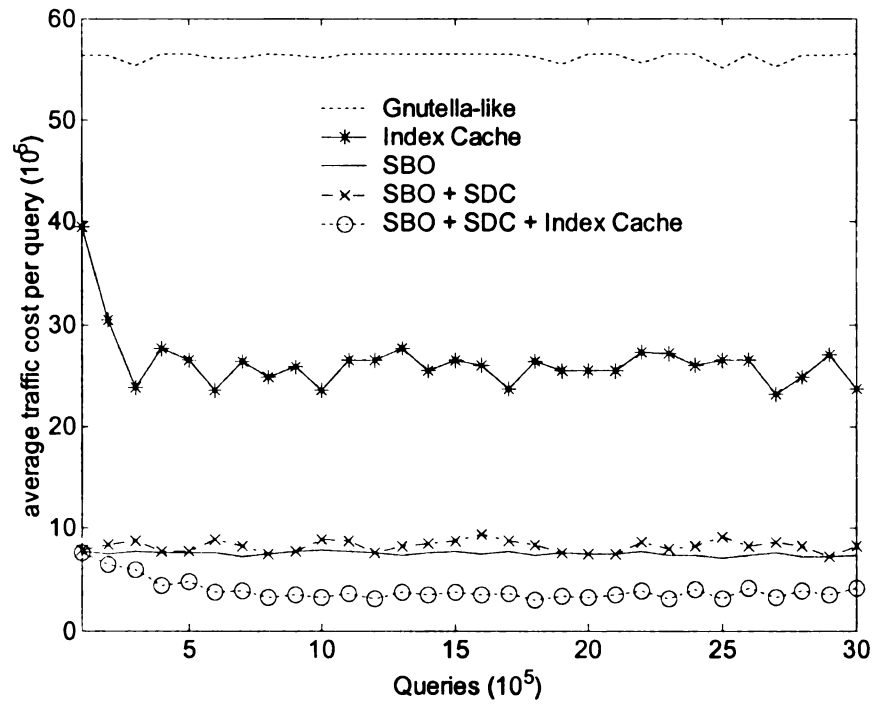
As we have mentioned, different values of average peer lifetime have been presented by previous studies [17, 72]. We further tune the average peer lifetime in our simulation. Figure 4.45 shows that SBO optimization operations can be conducted less frequently if average peer lifetime is longer.

From our simulation results, we find that if the average peer lifetime is longer than 37 minutes, the event-driven policy will outperform periodic policy.

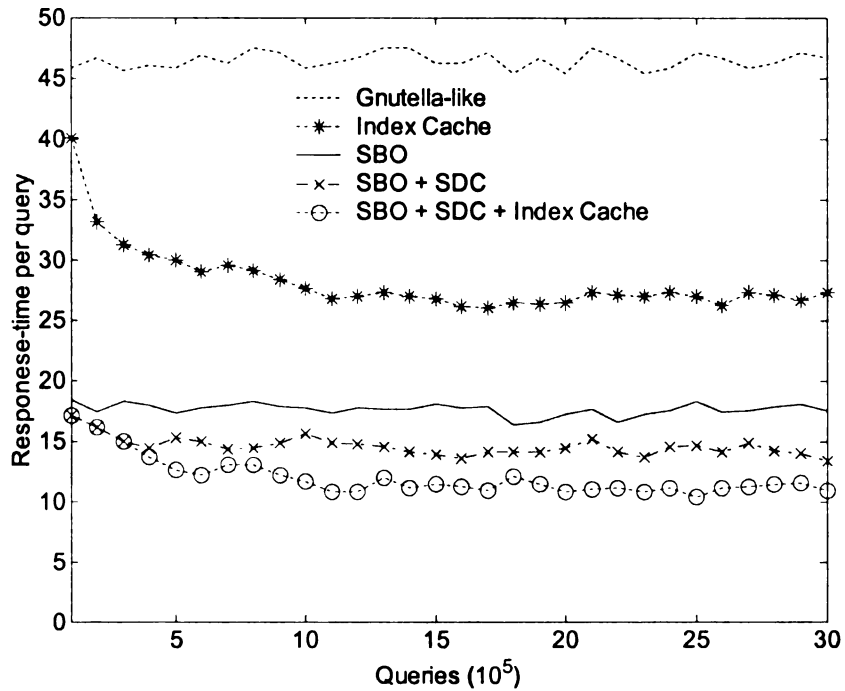
In a super peer P2P system, such as KaZaA, flooding based search is only employed among super peers. The mechanism to select super peers makes the super peers more stable than leaf peers. Thus, an event-driven policy is highly recommended when SBO is implemented among super peers.

#### **4.4.7 SBO with SDC and Index Caching**

We have discussed the design of SDC to further improve SBO. In this part, we evaluate SDC based on SBO and a strategy of combining SBO with response index caching scheme. We compare the traffic cost with SDC enabled SBO optimization plus response index caching in Figures 4.46 and 4.47. The design of SDC can further improve average response time of SBO by about 25% with very trivial traffic cost increment. Also compared with SBO, by combining SDC enabled SBO with response index caching the traffic cost is reduced by about 50% without shrinking the search scope, and the average query response time is reduced by about 42%.



**Figure 4.46 Traffic cost reduction of SBO in dynamic P2P environments**



**Figure 4.47 Response time reduction of SBO in dynamic P2P environments**

## 4.5 Two Hop Away Neighbor Comparison and Selection (THANCS)

Two Hop Away Neighbor Comparison and Selection (THANCS) scheme effectively attacks the topology mismatch problem and optimizes the overlay topology to approach the optimal solution. In THANCS, each peer probes the distances with its immediate logical neighbors and piggybacks all its neighbors' distance information with selected query messages. Thus, peers may have two hop away neighbors' information without much extra overhead to optimize the topology to approach an optimal overlay. THANCS consists of two main components: piggybacking neighbor cost on queries, and neighbor comparison and selection.

### 4.5.1 Piggyback Neighbor Distance on Queries

We use network delay between two peers as a metric for measuring the distance between the peers. In THANCS, each peer probes the distances with its immediate logical neighbors and stores the information in its local storage. Peers in Gnutella have a limited number of neighbors, 4 to 6 on average, so the overhead of this operation is trivial for each peer. Since Internet paths are relatively stable [95], to keep the neighbor information up to date, a peer only needs to probe the distance to its new neighbor and modify the neighbor distance information when any of its neighbors is leaving.

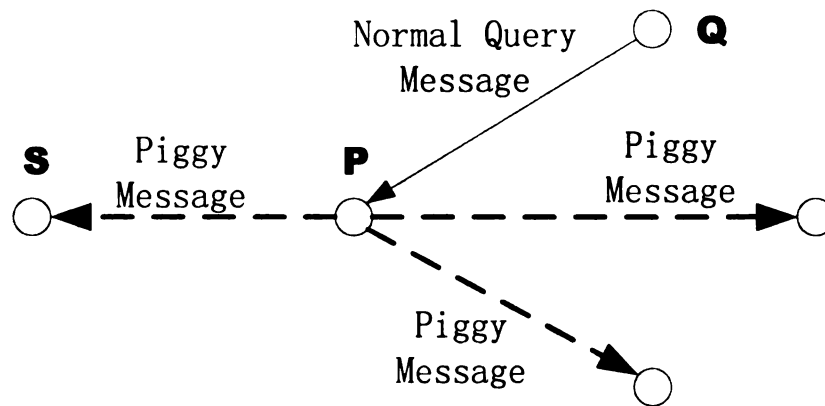
**Table 2: Piggy Message body**

	Neighbor IP Address	Neighbor Distance
Byte offset	0                      3	4                      5

For each peer to keep the distance information of its neighbors, we add one special query message type, Piggy Message. A piggy message has a message body in the format as shown in Table 2. It includes two fields: Neighbor's IP Address and Neighbor's Dis-

tance. Since a piggy message will be piggybacked by a query message, it does not need the Gnutella's unified 23-byte header.

A peer constructs a piggy message for each of its neighbors. The idea here is to send the piggy message of one neighbor to all the other neighbors. In order not to increase the number of messages, THANCS is designed to piggyback a piggy message on a query message.

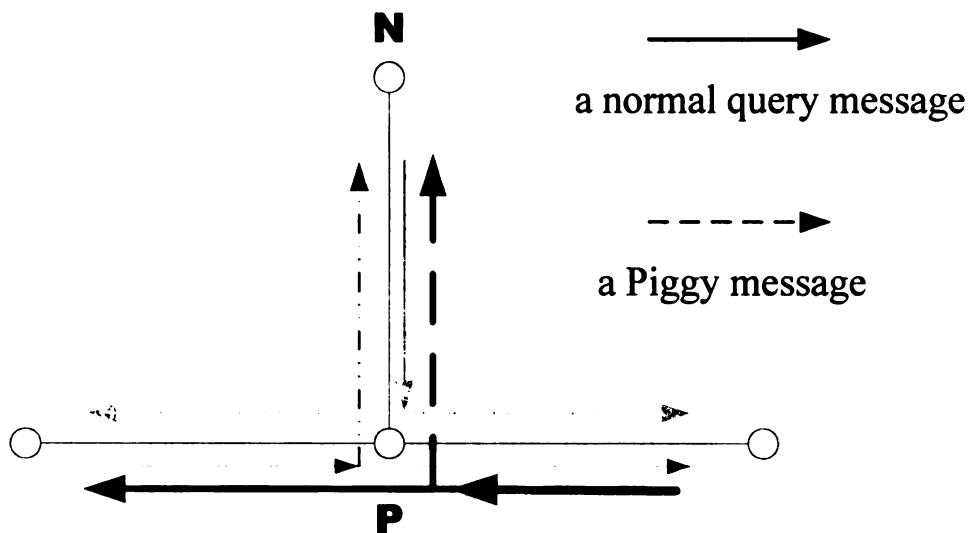


**Figure 4.48 Forwarding piggy messages**

For example, peer P in Figure 4.48 constructs a piggy message for its neighbor Q, which contains Q's IP address and the distance of PQ. When P receives a query from Q, this piggy message will be piggybacked by the query message that will be forwarded to all the other neighbors of peer P. The payload type of a query message piggybacking a piggy message can be defined as 0x82 to distinguish it from a normal query message (0x80). The payload length of such a query message will be increased by 6. After receiving such a query message, each of the other neighbors will detach the piggy message from the query message, record the distance information of P to Q, and further process the query and forward the query message if necessary. The piggybacked piggy message

will not be forwarded, but it is possible for this query message to piggyback another piggy message.

However, peer P may receive queries from Q very frequently. It is obviously not necessary for all query messages from Q to piggyback the piggy message. One critical issue to be examined here is which incoming queries should piggyback the piggy message. Although a piggy message is only 6-byte long, a large amount of duplicated piggy messages still inserts a lot of unnecessary traffic into the network. In this study we present two policies for this selection: pure probability-based (PPB) policy and new neighbor triggered (NNT) policy.



**Figure 4.49 New neighbor triggered ploicy**

The PPB policy is simply providing a pre-defined probability,  $\alpha$ , for a query to decide to piggyback a piggy message. That means in each peer, the probability for a query to piggyback a piggy message is  $\alpha$ . A smaller  $\alpha$  means there will be fewer queries piggybacking piggy messages. Note that in this design, it is not to say that once a query starts to piggyback a piggy message, the piggy message will be forwarded with the query mes-

sage until the query message is dropped. Instead, each piggy message will be piggybacked for only one single hop. The piggy message will be detached from the query message from the previous peer. With the probability  $\alpha$ , this query message may piggyback another piggy message in the current peer. The major advantage of this policy is its simplicity.

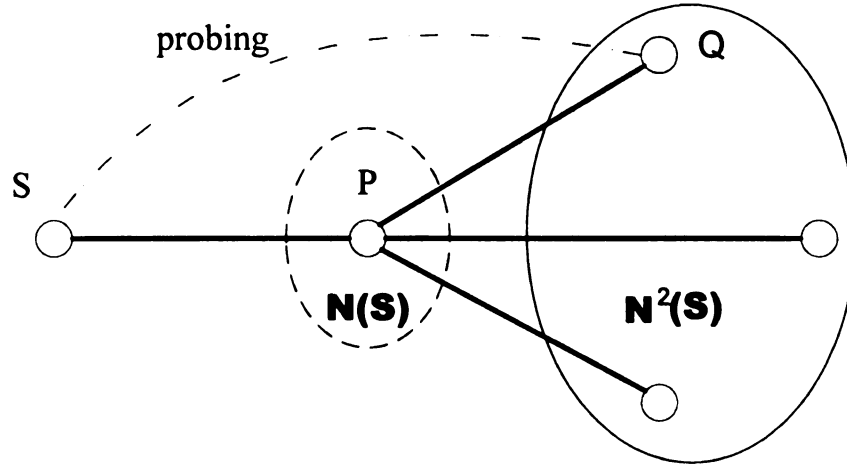
In the NNT policy, query messages will not piggyback piggy messages until a peer finds a neighbor who just joined the P2P network. As shown in Figure 4.49, all peers monitor new neighbors' coming. For example, when a peer P gets a new neighbor N, P does the following two operations. First, peer P probes the distance with N and constructs a piggy message. Peer P lets the first query message coming from N piggyback this piggy message. This query message with piggybacked piggy message will be forwarded to all P's existing logical neighbors except peer N. Second, the first incoming query from each of P's existing neighbors after peer N's coming will piggyback a corresponding piggy message (with the distance to this neighbor) when it is forwarded to N. However, the query message will not piggyback a piggy message when it is forwarded to the other existing neighbors, as illustrated in Figure 4.49. Another option is to let the first query message from P's previous neighbors piggyback all the piggy messages (except the one for the new neighbor) to the new neighbor, N. Compared with the PPB policy, NNT policy is relatively complicated while it has smaller traffic overhead.

We will further discuss the selection of the probability for this policy, and the selection of these two policies in detail later.

S will know the distance between each peer in  $N(S)$  and any of the peers in  $N^2(S)$  that are connected with the peer in  $N(S)$ . This information will be used in the second component, neighbor comparison and selection.

#### 4.5.2 Neighbor Comparison and Selection

The behavior of THNACS peers in this neighbor comparison and selection component is demonstrated through an example in Figure 4.50. An arbitrary peer, S, probes the distance to all the known un-probed  $N^2(S)$  peers. The distance of SP is known to S. When peer S receives a piggy message from peer P with the distance of PQ, there will be two cases.



**Figure 4.50 Probing two hop away neighbors**

*Case 1:* Peer Q is also a direct neighbor of peer S, i.e.  $Q \in (N(S) \cap N^2(S))$ . In this case, peer S will compare the cost of SQ, SP, and PQ. If SQ or SP is the longest in these three connections, S will put the longest connection into its will-cut list that is a list of connections to be cut later, e.g. 50 second later. If PQ is the longest, peer S will do nothing because the system is fully distributed and peer P or Q will disconnect PQ shortly. A peer

will not send or forward queries to connections in its will-cut list, but these connections have not been cut in order for query responses to be delivered to the source peer along the inverse search path.

*Case 2:*  $Q$  is a two-hop away neighbor of  $S$ , but not a direct logical neighbor of  $S$ , i.e.  $Q \in (N^2(S) - N^1(S))$ . Peer  $S$  will first check whether it had probed peer  $Q$  before or used to have peer  $Q$  as a direct neighbor by looking up  $S$ 's distance-cache that is designed to keep a list of peers that have been probed by peer  $S$ . If peer  $S$  used to probe the distance to peer  $Q$ ,  $S$  will do nothing with peer  $Q$  and start probing other peers in  $N^2(S)$ . Otherwise, peer  $S$  will probe the distance to peer  $Q$ , and store the probing result in the distance-cache. Having the distance of  $SQ$ , peer  $S$  compares  $SQ$ ,  $SP$ , and  $PQ$ . If  $SQ$  is the longest, peer  $S$  will not keep the connection with peer  $Q$ . If  $SP$  is the longest,  $S$  will keep the connection with peer  $Q$  and put  $SP$  into the will-cut list. If  $PQ$  is the longest in the three connections,  $S$  will keep the connection with both  $P$  and  $Q$ , expecting that peer  $P$  or  $Q$  will disconnect  $PQ$  later.

The first component, piggyback neighbor distance on queries, is relatively straightforward. The selection of PPB or NNT policy will be further discussed in Section 6. Thus, we do not provide the detailed pseudo code for this part here. The pseudo code of neighbor comparison and selection component for a given source peer  $i$  is as below. Let  $D_{ij}$  represent the distance from peer  $i$  to  $j$ .

**Pseudo code of neighbor comparison and selection:**

**for** each  $h \in N^2(i)$  in  $j$ 's direct neighbors,

**if**  $h \in N(i)$

    {check  $D_{ih}$ ;

```

if  $(D_{ih} > D_{ij}) \cap (D_{ih} > D_{jh})$ 
    {put connection  $ih$  into  $i$ 's will-cut list; }

else if  $(D_{ij} > D_{ih}) \cap (D_{ij} > D_{jh})$ 
    {put connection  $ij$  into  $i$ 's will-cut list; }

end if

else
    if ( $D_{ih}$  is not in  $i$ 's distance-cache)
        Probe  $D_{ih}$ ;

        if  $(D_{ih} < D_{ij}) \cup (D_{ih} < D_{jh})$ 
            {keep the connection with peer  $h$  and include  $h$  as a direct logical
            neighbor of  $i$ ;

            if  $(D_{ij} > D_{ih}) \cap (D_{ij} > D_{jh})$ 
                {put connection  $ij$  into  $i$ 's will-cut list; }
            }

        end if
    end if

end if

end for

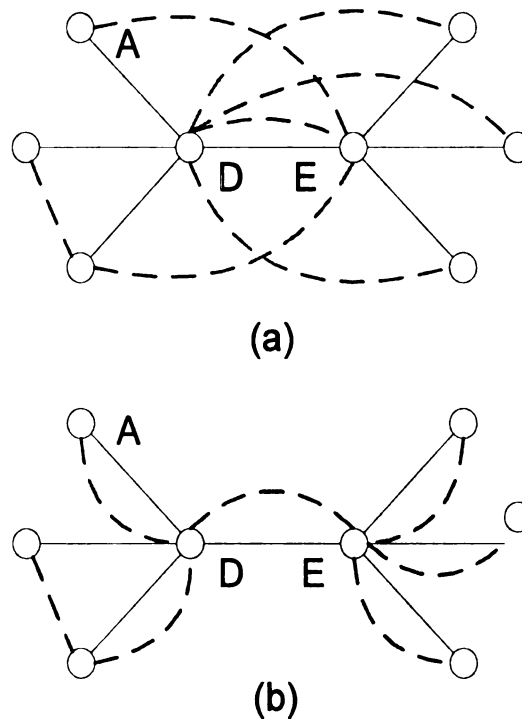
```

#### 4.5.3 Effectiveness Analysis of THANCS

To the best of our knowledge, the work in [69] is the first to mention the topology mismatch problem in Gnutella like P2P systems and comprehensively discuss the importance of a “proper fitting” overlay topology. The authors also provided an example to illustrate this problem, as shown in Figure 4.51. In this figure, solid lines represent the un-

derlying infrastructure that connects the eight hosts in a Gnutella-like P2P system, and dotted lines denote the overlay connections.

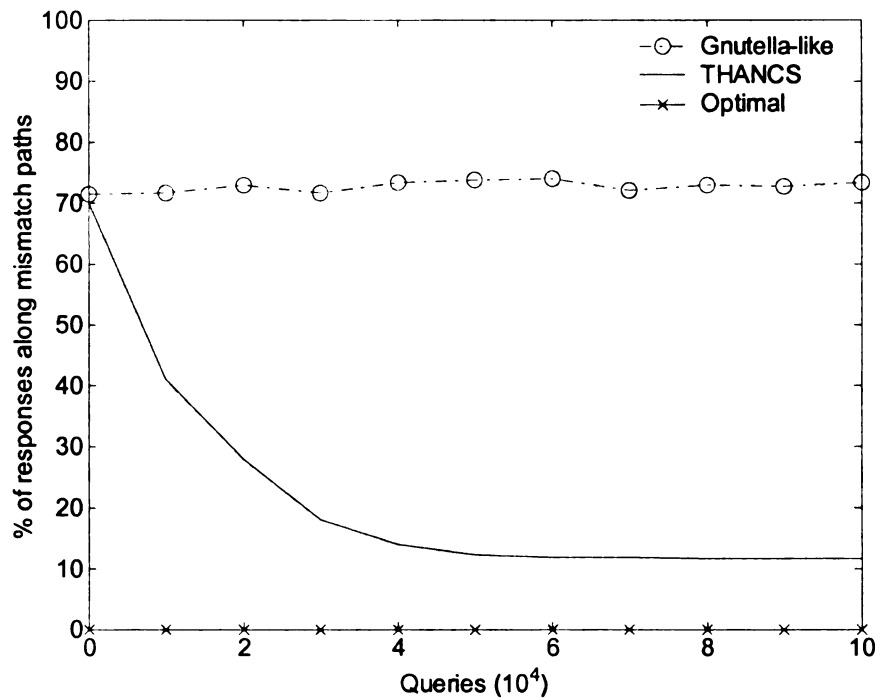
In an inefficient overlay shown in Figure 4.51(a), a message from node A involves six communications over the physical link D-E. This inefficient overlay can be optimized by THANCS. For example, when peer A receives peer E's piggy message indicating the cost of D-E, A will create overlay connection A-D and disconnect connection A-E shortly. After one or two steps of THANCS optimization, the overlay shown in Figure 4.51(a) will be optimized as the overlay shown in Figure 4.51(b), in which a message from node A involves only one communication over the physical link D-E.



**Figure 4.51** THANCS can effectively optimize a mismatched overlay topology to a better mapping overlay. (a) With inefficient mapping, a broadcast message issued by node A travels physical link D-E six times. (b) After THANCS optimization, with a better mapping, the same message traverses D-E link only once.

#### 4.5.4 Effectiveness of THANCS in Static Environments

We first study the effectiveness of THANCS in a static, so that we may contrast the effectiveness of THANCS optimization with an optimal overlay algorithm. We have proven that the DMAD problem is NP-hard, so this simulation is conducted based on a small size overlay topology. Specifically, in this simulation, 128 out of 27,000 nodes are randomly selected as peering nodes. We then generate 100,000 queries, and simulate flooding search for different overlay topologies including a Gnutella-like overlay, a THANCS optimized overlay, and an optimal overlay that is obtained by a brute force algorithm. Here we have two assumptions. First, we assume that the underlying physical layer uses shortest path routing with delay as the metric. Second, we assume the average number of neighboring peers is 4, or there are totally 256 overlay connections in the 128-peer topology. We run the simulation 20 times with different random choices of peers and report the average results of 20 runs.



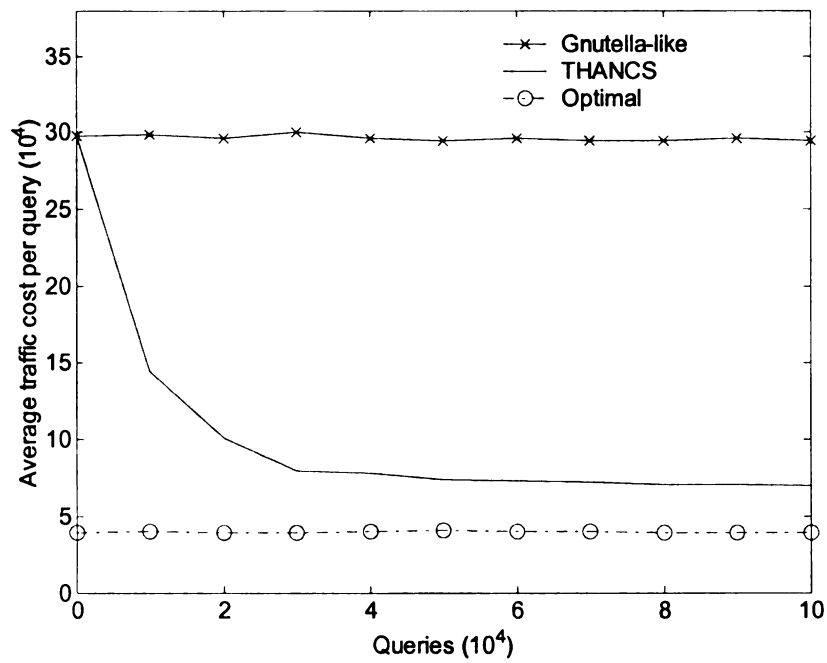
**Figure 4.52 The percent of query responses along mismatched paths of three schemes**

Our simulation results show that in an optimal overlay, no response comes back along mismatched paths. In Figure 4.52 we show the percentage of query responses along mismatched paths of three schemes. Recall that 70% of the queries are back along mismatched paths in Gnutella like P2Ps. An optimal overlay effectively replaces all the mismatched paths as shown in Figure 4.52. THANCS is proven an effective approach optimizing up to 58% out of the 70% mismatched paths. As a result, system performance is improved significantly. Figure 4.53 shows that the traffic cost,  $c$ , decreases by up to 75% when optimization operations of THANCS are conducted. In Figure 4.54 we show that THANCS can effectively shorten the query response time by about 60%.

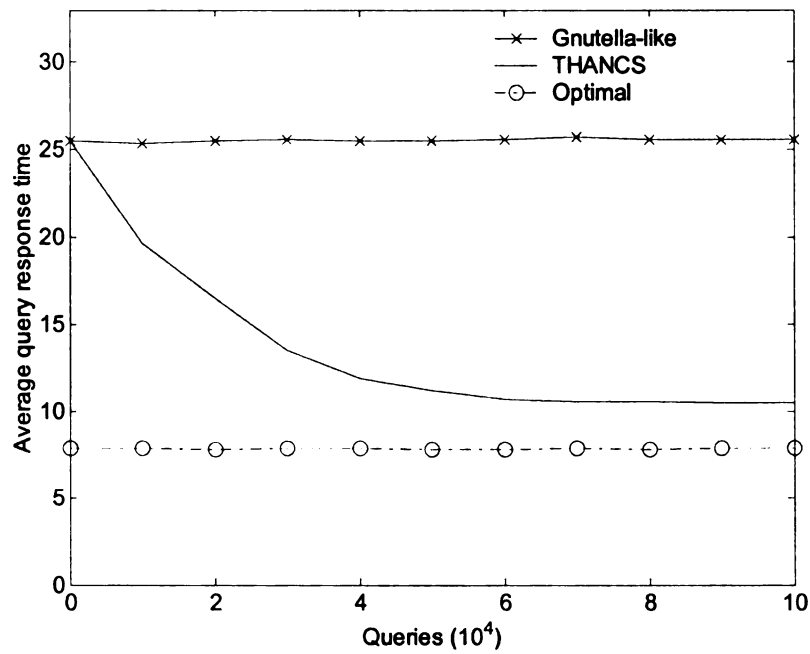
We also show the search efficiency on the optimal overlay in these two figures. Although there is no bound indicating how well THANCS could do, the simulation results show the overall performance of THANCS is close to the optimal solution in a static environment.

#### **4.5.5 THANCS in Dynamic Environments**

Previous simulations are based on a static P2P environment on top of a small size overlay and we have mentioned that the difficulties of solving topology mismatch problem are that peers are randomly coming and leaving, and real P2P systems have a huge number of online users. We further evaluate the performance of THANCS in a dynamic environment with a large number of peers. We first discuss how to select a better policy from PPB and NNT. We then evaluate the effectiveness of THANCS by varying the size of the P2P system and the average number of connections in the overlay network.



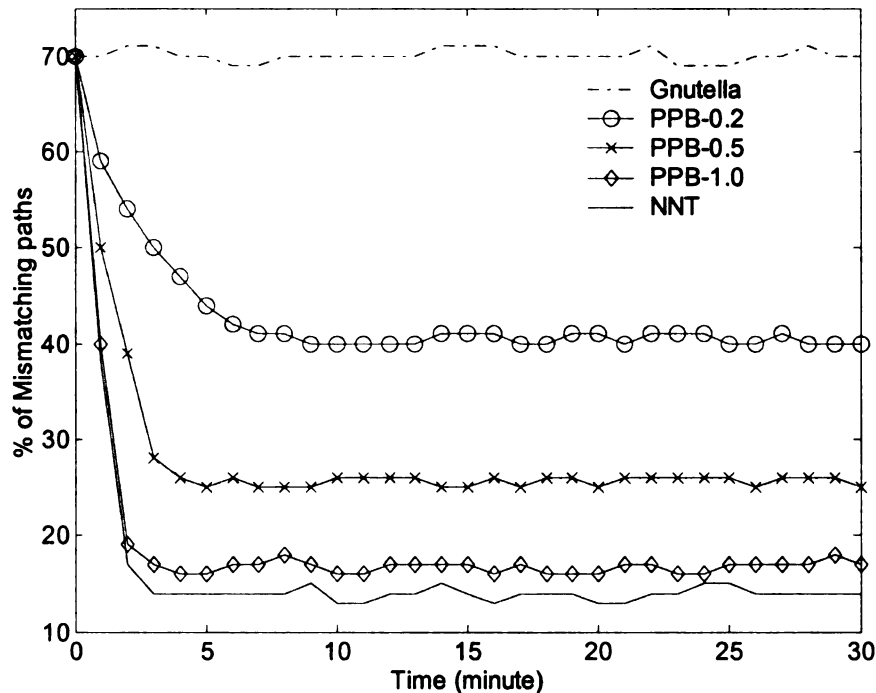
**Figure 4.53 Average traffic cost per query**



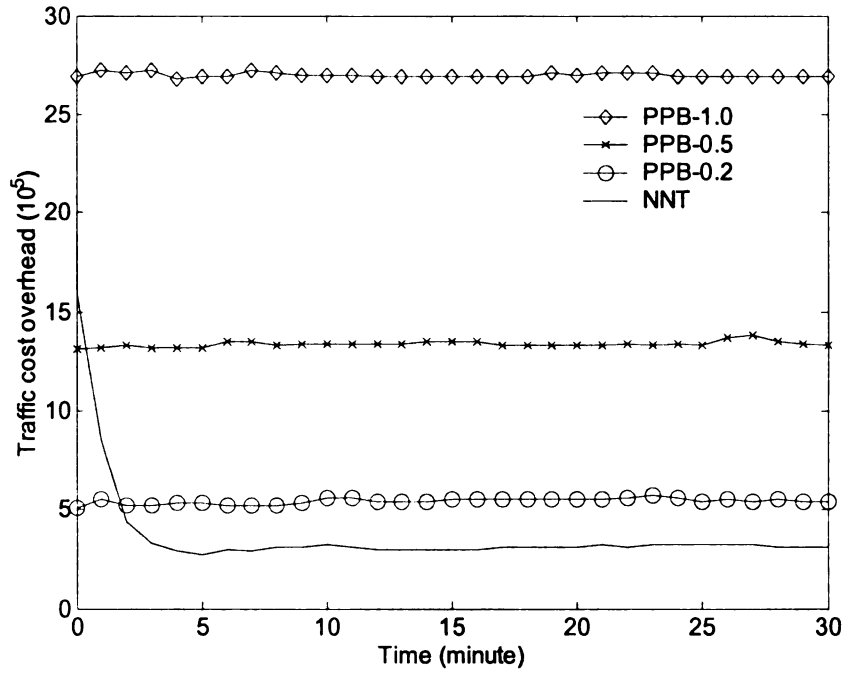
**Figure 4.54 Average query response time**

THANCS is a completely distributed approach, so each single peer determines its own operations independently without any help from a central server. One key issue for peers is which query message should be piggybacked with a piggy message.

We have discussed two options, PPB and NNT. We evaluate these two policies here. In this part of the simulation, we select 5,000 nodes from the 27,000 nodes physical topology as overlay peers and simulate flooding based search for 50,000 queries, which means around 30 minutes in a real P2P environment. Figure 4.55 plots the performance of THANCS on solving the topology mismatch problem when using NNT policy and PPB policy. The curve of 'PPB- $\alpha$ ' shows the performance of PPB, where the probability of an incoming query to piggyback a piggy message is  $\alpha\%$ . Figure 4.56 plots the traffic overhead of these two policies.



**Figure 4.55 Performance of THANCS with NNT or PPB policies on reducing mismatch degree**



**Figure 4.56 Traffic cost overhead of THANCS with NNT or PPB policies**

The most attractive advantage of PPB policy is its simplicity. In PPB policy, each peer blindly selects some of the queries using a straightforward rule to piggyback piggy messages without monitoring logical neighbors' coming or leaving. Indeed, the overhead of PPB is not intolerable, although it is high as shown in Figure 15. In this simulation, the mismatching reduction of PPB-1.0 is close to NNT policy, but PPB-1.0 has the largest traffic overhead. This overhead in the simulated environment, however, is only equivalent to the traffic of a couple of queries per minute, accounting for less than 2% of the traffic savings by THANCS. The NNT policy can further reduce optimization traffic overhead, but it is a bit more complicated compared with PPB policy. Simulation results in Figures 4.55 and 4.56 show that NNT outperforms PPB in the sense that it has a higher optimization rate and a smaller traffic overhead. Thus, we adopt NNT policy. The traffic overhead of NNT is less than 0.3% of the traffic savings.

To better evaluate the performance of THANCS in dynamic environments, we vary the size of the Gnutella-like overlay networks from 2,000 to 8,000. We simulated the search process on each of the overlays for 30 minutes, and repeat this simulation with different random seeds for 20 times. We plot the average results with and without the THANCS optimization scheme. Here we use two metrics, traffic cost reduction rate ( $R_c(*)$ ) and response time reduction rate ( $R_t(*)$ ).  $R_c(*)$  is defined by:

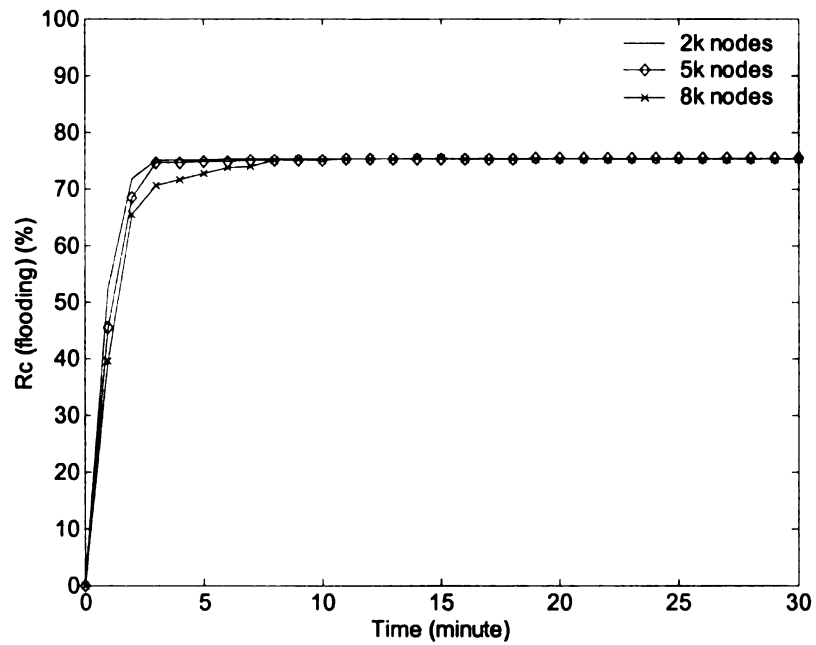
$$R_c(*) = \frac{C(\text{Gnutella-like}) - C(\text{THNACS})}{C(\text{Gnutella-like})} \times 100\%$$

where  $C(\text{Gnutella-like})$  represents the traffic cost incurred by searching all the peers using the given mechanism (\*), such as blind flooding, in a Gnutella-like overlay.  $C(\text{THNACS})$  represents the traffic cost incurred in THANCS enabled P2P networks.

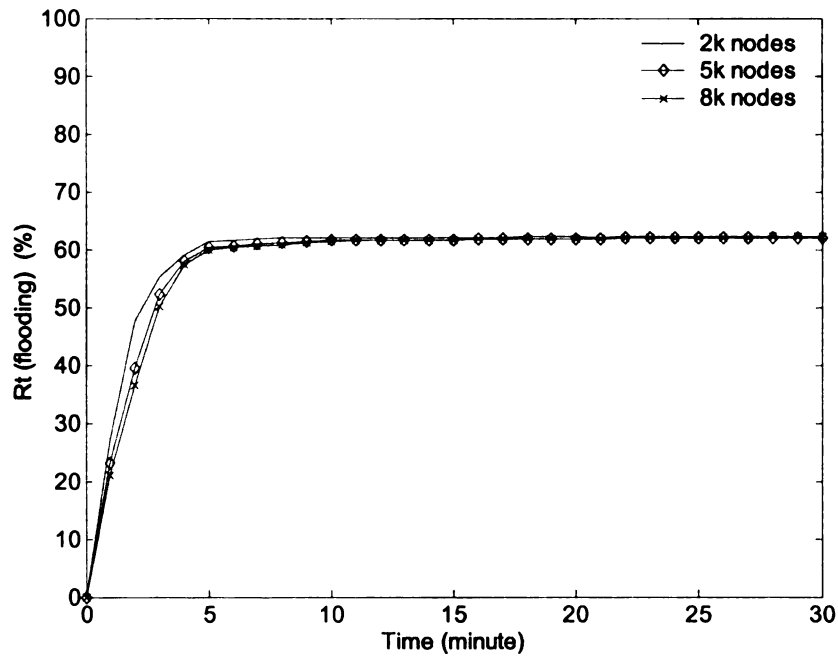
$R_t(*)$  is given by:

$$R_t(*) = \frac{T(\text{Gnutella-like}) - T(\text{THNACS})}{T(\text{Gnutella-like})} \times 100\%$$

where  $T(\text{Gnutella-like})$  denotes the average response time of queries in Gnutella-like overlays, and  $T(\text{THNACS})$  is that of THANCS enabled systems. In this simulation, we study the performance of THANCS based on the blind flooding mechanism, and we will demonstrate the effectiveness of THANCS on other existing advanced search strategies shortly. The variance of data in different simulation runs is very small, so we do not show the confidence intervals in the figures. Figures 4.57 and 4.58 show that the performance of THANCS is not strongly dependent on the size of the network.



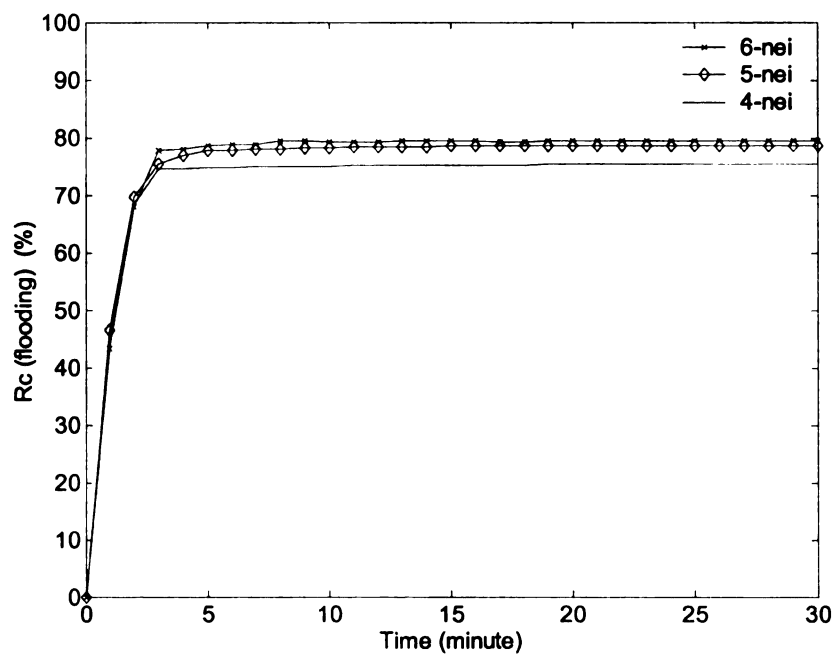
**Figure 4.57 Traffic cost reduction of THANCS in different size overlays ranging from 2,000 to 8,000 nodes**



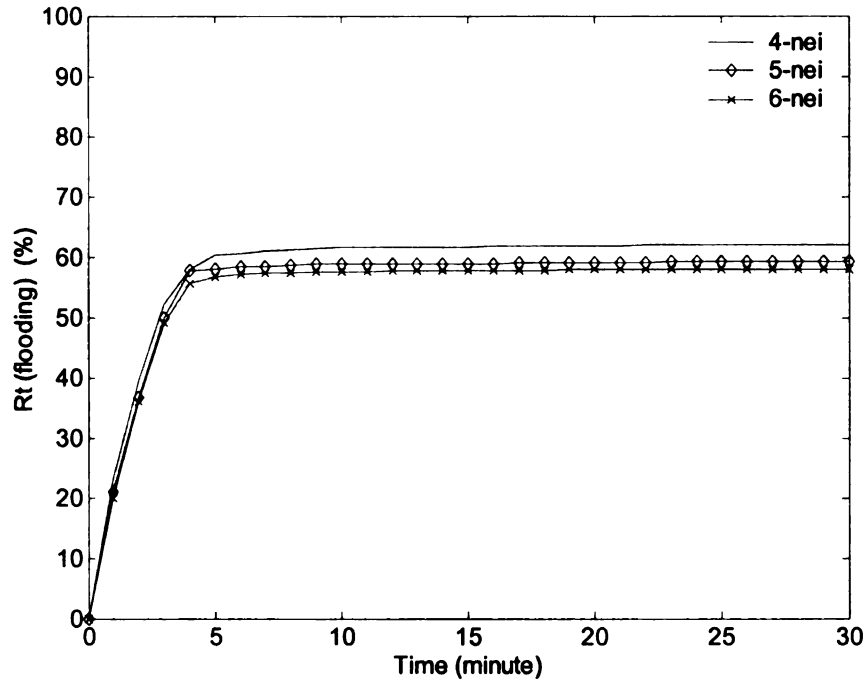
**Figure 4.58 Average response time reduction of THANCS in different size overlays ranging from 2,000 to 8,000 nodes**

With THANCS enabled,  $R_c$  (flooding) is up to approximately 75%, which means that the total network traffic incurred by blind flooding search will be decreased by 75% without shrinking the search scope. At the same time, average query response time is also decreased by approximately 60%. We also tried THANCS in physical topologies based on real AS-level Internet topologies [13] and the results proved consistent with those of generated topologies.

We demonstrate the effectiveness of THANCS by varying the average number of logical neighbors from 4 to 6, and plot the results in Figures 4.59 and 4.60. As shown in Figure 4.59,  $R_c$  (flooding) is higher than 75% when peers have 4 neighbors, and it grows slightly when average connection number is increased. Figure 4.60 shows that the reduction in response time drops slightly when peers have more edges, but still at a level of higher than 55%.



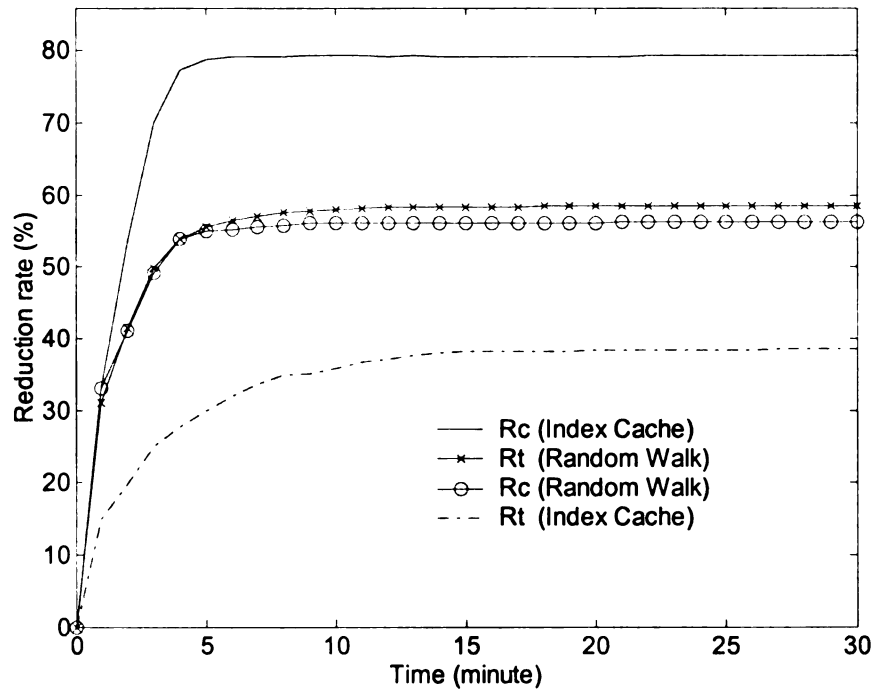
**Figure 4.59 Traffic cost reduction of THANCS in 5,000-node overlays with different average number of neighbors ranging from 4 to 6**



**Figure 4.60 Average response time reduction of THANCS In 5,000-node overlays with different average number of neighbors ranging from 4 to 6**

#### 4.5.6 Effectiveness with other advanced search strategies

We have run simulations under the blind flooding search mechanism based on the following observations. First, although many advanced search mechanisms are proposed, flooding based search is still the most popular search mechanism widely used in today's real systems. Second, topology optimization based approaches are orthogonal with other existing search mechanisms, such as forwarding based or cache based optimization. THANCS could be combined with other types of advanced search strategies and further improve search efficiency of P2P systems. Therefore, the effectiveness of THANCS on flooding based search can also reflect the effectiveness of THANCS on other search mechanisms.



**Figure 4.61 Traffic cost and average response time reduction on Random Walk and Index Cache schemes when employing THANCS**

In our simulation, we have designed and built some schemes that combine THANCS with two popular advanced search mechanisms, Random Walk, and Index Caching. We first simulate search under a simple random walk scheme, as introduced in [55]. For each query we issue 30 walkers. We then combine THANCS with Index Caching, in which query responses are cached in passing peers along the returning path [82]. Each peer keeps a local cache and a response index cache. The size of a response index cache is limited to 200 items.

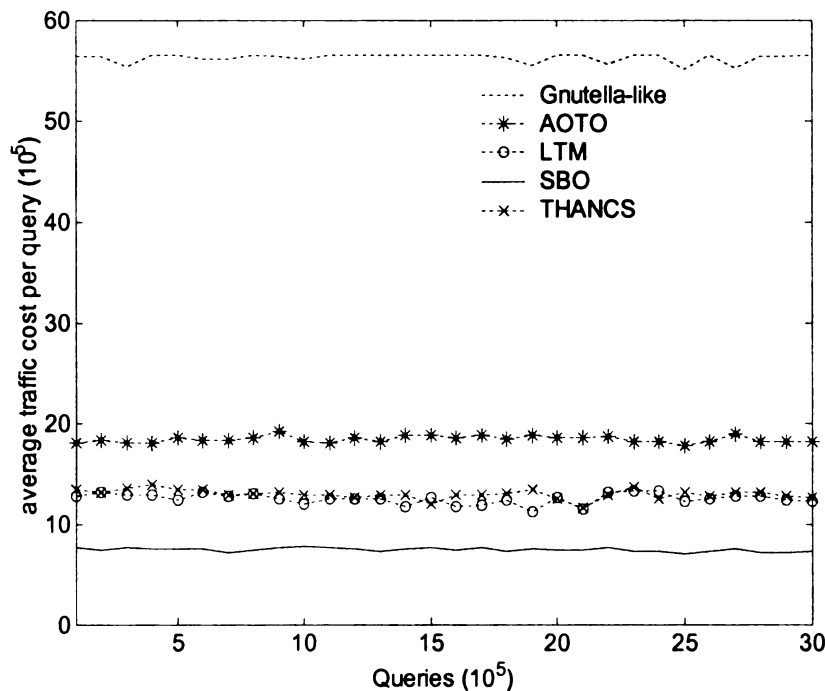
We can see in Figure 4.61 that THANCS can further improve search efficiency when it is employed with these advanced approaches. In Figure 4.61, we show Rc (Random Walk), Rt (Random Walk), Rc (Index Cache), and Rt (Index Cache). As expected, THANCS reduces traffic cost and response time of the Index caching scheme by ap-

proximately 80% and 39% respectively, and reduces both traffic cost and response time of the Random Walk by more than 55%.

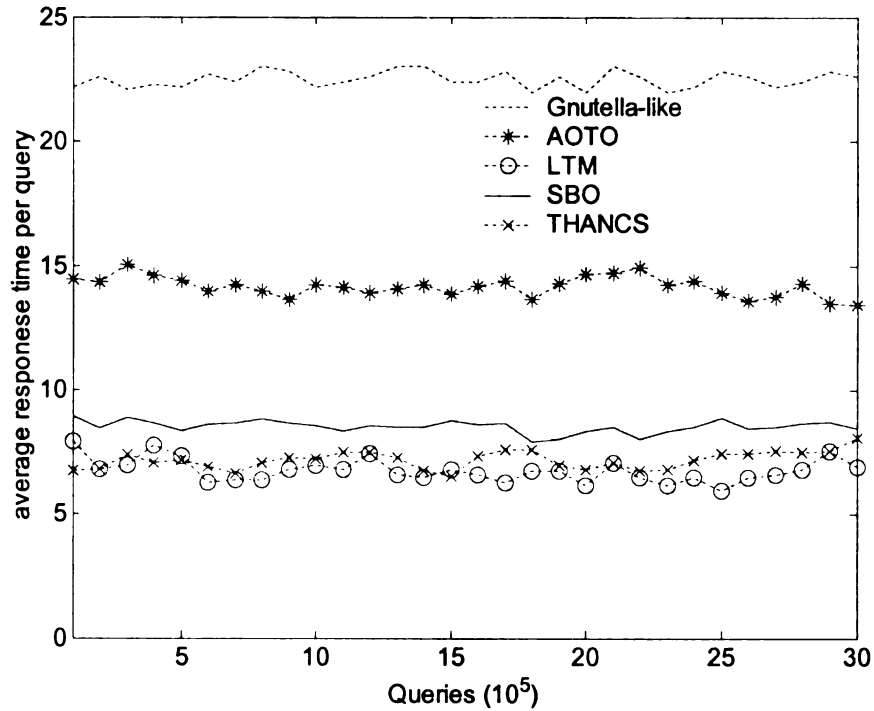
## 4.6 Discussion

Without assuming any knowledge of the underlying physical topology, the conventional P2P mechanisms are designed to randomly choose logical neighbors, which cause serious topology mismatch problems between the P2P overlay network and the underlying physical network.

To alleviate the mismatch problem and reduce the unnecessary traffic and response time, four schemes are introduced in this chapter: Adaptive Overlay Topology Optimization (AOTO), Location-aware Topology Matching (LTM), Scalable Bipartite Overlay (SBO), and Two Hop Away Neighbor Comparison and Selection (THANCS).



**Figure 4.62 Traffic cost reduction of AOTO, LTM, SBO and THANCS**



**Figure 4.63 Response time reduction of AOTO, LTM, SBO and THANCS**

All of them achieve the above goals without bringing any noticeable extra overheads. Moreover, these techniques are scalable because the P2P overlay networks are constructed in a fully distributed manner where global knowledge of the network is not necessary.

We compare the performance of these four approaches in dynamic P2P environments in Figures 4.62 and 4.63. In this simulation, the size of overlay topology is 5,000, the average neighbor number is 6, and the physical topology has 27,000 nodes.

From the figures we can see that, although AOTO is the simplest one, its convergent speed is the slowest, so its overall performance in dynamic environments is not as good as the other three approaches. The convergent speed of LTM is the fastest, but it needs the support of NTP [14] to synchronize the peering nodes. Since THANCS has very simi-

lar performance with LTM, but does not need to synchronize peers. SBO, incurring only half overhead of AOTO, reduces the traffic cost the most. Comparing THANCS and SBO, SBO has less overhead and THANCS has lower response time, which is because THANCS converges faster than SBO. Thus THANCS performs better in a more dynamic environment. When the peers are frequently coming and leaving the system, such as a non-super peer Gnutella-like system, we prefer THANCS. In a more stable system, such as the super peer layers in KaZaA, using SBO is a good choice.

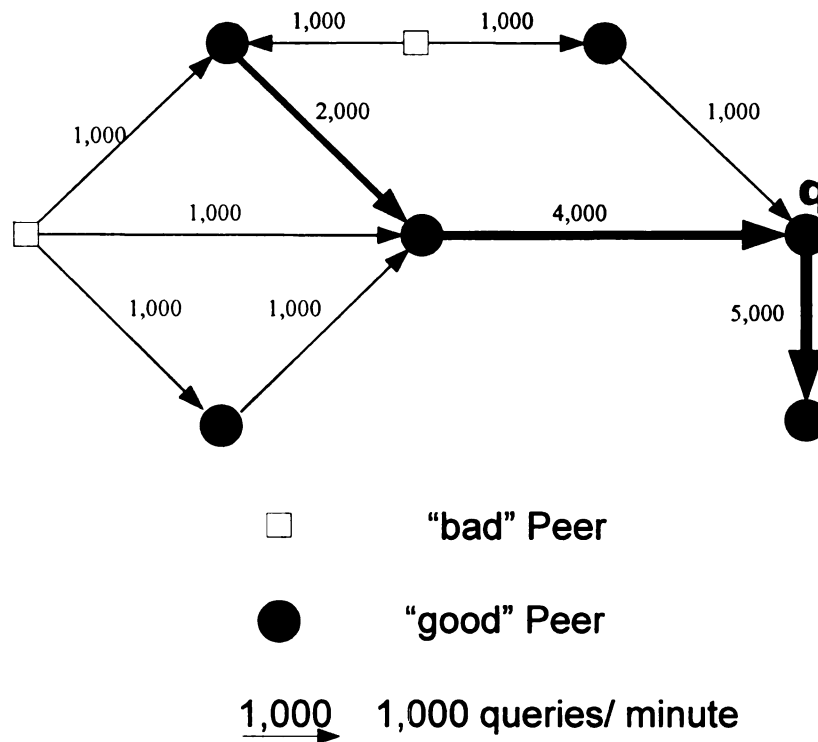
## 5 Defending P2Ps from Overlay Flooding-based DDoS

Our belief that unstructured P2P systems, such as Gnutella- and KaZaA-like networks, are vulnerable to overlay DDoS attacks is based on following observations.

First, the flooding based search mechanism makes overlay DDoS in P2Ps simple in design and operation without requiring any special resources. Malicious nodes may attack the system by simply walking in and sending out a huge number of useless search queries. The message volume will be quickly propagated so that the resources in the P2P system can be exhausted by a small number of DDoS attack machines.

Second, the anonymity design of P2P helps the malicious nodes easily hide behind other peers. For example, the forwarding peers do not know the original sender of each query and the query response is only delivered to the neighbor along the inverse path of the search path. Attempting to disconnect all the peers who send out a large number of queries is dangerous in that “good” peers could be forwarding queries for “bad” peers. We show a simple example in Figure 5.1. Instead of flooding the same queries to all its neighbors, a “bad” peer issues different queries to its neighboring peers in order to make DDoS attacks more damaging to the system. We can see in Figure 5.1 that the query traf-

fic in some connections between “good” peers is much higher than that in the connections between a “bad” peer and a “good” peer.



**Figure 5.1 Although peer q sends out 5,000 queries per minute, it is a “good” peer**

Finally, the nature of free downloading in P2P systems makes peers easily recruited as DDoS agent (slave) peers by downloading files with special packets attached.

It is challenging to handle overlay DDoS in P2P systems because of the extreme difficulty in separating attack query traffic from legitimate traffic, while it is relatively easy to perform DDoS in P2P without being detected. Note that the victim of P2P overlay DDoS is not a single peer but the P2P system itself.

In this chapter, I introduce a detection based approach, DD-POLICE (**Defending P2Ps from Overlay Distributed-Denial-of-Service**), to protect P2P systems against overlay DDoS. In DD-POLICE, peers monitor the traffic from and to each of their neighbors. If a

peer receives a very large number of queries from one of its neighbors, it will mark this neighbor as a suspicious DDoS peer. This peer will exchange query volume information with the suspicious peer's neighbors so that it can figure out the number of queries originally issued by the suspicious peer. Based on this number, this peer makes a final decision on whether the suspicious neighbor is a DDoS peer and should be disconnected.

### ***5.1 Definition of a “good” Peer and a “bad” Peer***

In our model, the definition of a “good” peer is twofold. First, we assume they have the same processing capacity, and they will forward as many incoming queries as they are capable of. Second, a “good” peer will not issue more than 100 queries per minute. Actually, some observations in [82] show that one peer issues less than 1 query per minute on average. We also have done some experiments in our lab, and no peer ever created more than 40 queries per minute. The assumption that a “good” peer does not issue more than 100 queries per minute makes sense as it is hard for a human user to generate more than one query every second.

We further model a “bad” peer's behavior pattern as it will do everything else as a “good” peer except that it generates and issues a large number of queries during every time unit, and sometimes even issues as many queries as it is able to automatically generate. Here we assume the overlay DDoS compromised peers will still forward queries because of the following observations: (1) simply not forwarding messages will not do much damage to the system because of the nature of query flooding, and (2) it is easy to enhance the protocol to detect peers refusing to forward any incoming queries and remove them.

We quantitatively define a “good” peer and a “bad” peer as follows. We use  $Q_{ih}(t)$  to denote the number of queries sent out (issued plus forwarded) from peer  $i$  to peer  $h$  during the time period from  $(t-1)^{th}$  to  $t^{th}$  time unit. We assume that a peer  $j$  has  $k$  neighboring peers,  $m_1, m_2 \dots m_k$ .

**Definition 5.1:** Peer  $j$ ’s General Indicator at time  $t$  is defined as:

$$g(j, t) = \frac{1}{qk} \left( \sum_{m=m_1}^{m_k} Q_{jm}(t) - (k-1) \sum_{m=m_1}^{m_k} Q_{mj}(t) \right),$$

where  $q$  is the threshold in distinguishing a “good” peer and a “bad” peer. We set  $q=100$  based on the above discussion that a “good” peer does not issue more than 100 queries per minute.

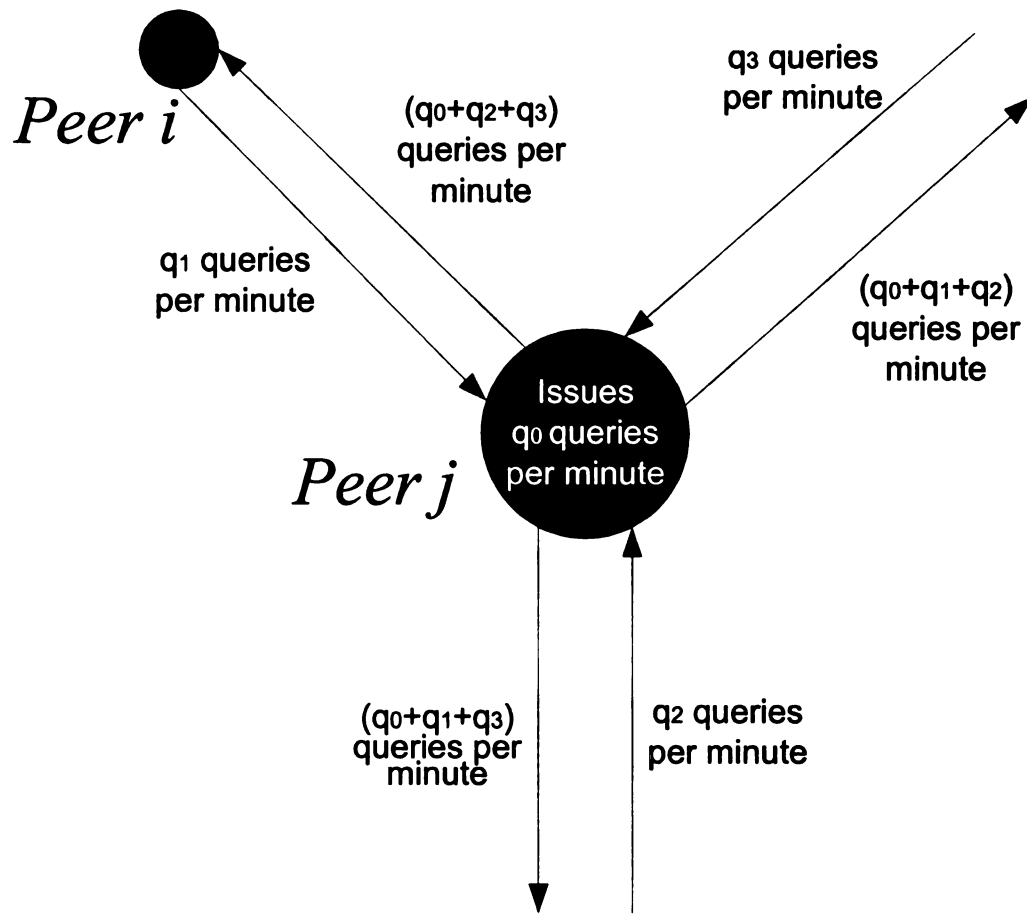
**Definition 5.2:** Peer  $j$ ’s Single Indicator measured by its neighboring peer  $i$  at time  $t$  is defined as  $s(j, t, i)$ , where

$$s(j, t, i) = \frac{1}{q} \left( Q_{ji}(t) - \sum_{\substack{m=m_1 \\ m \neq i}}^{m_k} Q_{mj}(t) \right)$$

**Definition 5.3:** For any peer  $j$  at any time  $t$ , for any peer  $i$  other than  $j$ , if  $g(j, t) > 1$  or  $s(j, t, i) > 1$ , peer  $j$  is a “bad” peer; otherwise, peer  $j$  is a “good” peer.

Figure 5.2 gives an example, in which peer  $j$  has three neighbors, i.e.  $k=3$ . At time  $t$ , peer  $j$  issues  $q_0$  queries and receives  $q_1, q_2, q_3$  queries from its three neighbors, respectively. From [9] we know that a query message will be dropped if the query message has visited the peer before. Thus, when peer  $j$  receives  $q_2$  and  $q_3$  queries from the other two neighbors, it may or may not send all  $q_0+q_2+q_3$  queries to its neighboring peer  $i$ , depending on the number of duplicated query messages from the other two neighbors. For sim-

plicity, here we assume there are no query message duplications in peer  $j$ , and all the incoming queries are sent out. This assumption is acceptable because  $q_0+q_2+q_3$  is an upper bound of the number of queries sent from peer  $j$  to  $i$ . Therefore, both  $g(j, t)$  and  $s(j, t, i)$  are less than or equal to  $q_0/q$ . If a peer's general indicator or single indicator is much larger than 1, we will have great confidence to disconnect it as a "bad" peer.



**Figure 5.2 Query traffic analysis**

To make this clear, let us compute the two indicators at peer  $i$  for peer  $j$  in Figure 5.2,

$$g(j, t) = \frac{1}{100 \times 3} ((q_0 + q_2 + q_3) + (q_0 + q_1 + q_2) + (q_0 + q_1 + q_3) - (3-1)(q_1 + q_2 + q_3)) = q_0 / 100$$

$$s(j, t, i) = \frac{1}{100} ((q_0 + q_2 + q_3) - (q_2 + q_3)) = q_0 / 100.$$

In this example  $g(j, t) = s(j, t, i) = q_0 / 100$ , where  $q_0$  is the number of queries issued by peer  $j$  per minute. Note that the assumption that all of peer  $j$ 's incoming queries should be forwarded is not always true. But at least we are sure that  $g(j, t)$  and  $s(j, t, i)$  could reflect  $q_0 / q$ .

## 5.2 *An Implementation of an Overlay DDoS Agent*

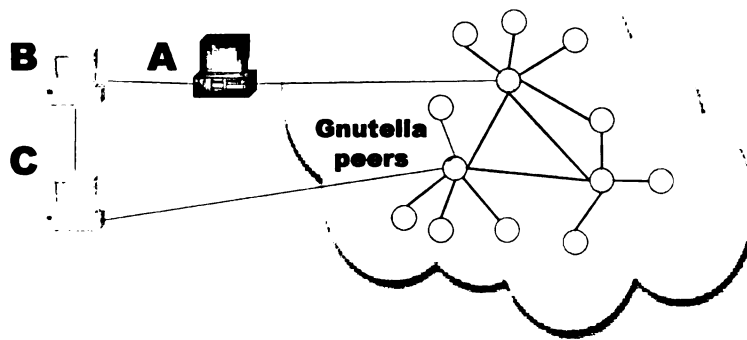
To investigate the impact of Overlay DDoS attack in P2P systems, we conduct a preliminary experiment in the following steps.

### 5.2.1 Query Trace Collection

We build a traffic-monitoring node to collect queries flooding through the Gnutella network. Using a modified LimeWire [11] client with logging functionality, all the queries passing by the monitoring node are recorded to a log file. The monitoring node is intentionally configured as a supernode connecting to ten peers in the Gnutella network, which makes the experimental traffic-monitoring node a "hot spot". Under this configuration, a large enough number of queries could be collected within a short time period.

The monitoring node was running on a PC with 2.4GHz Pentium IV processor and 100M Ethernet interface. Our experiment to collect query trace lasted 24 hours. We collected 13, 705, 339 queries with the size of 112 MB.

### 5.2.2 An Implementation of an Overlay DDoS Agent



**Figure 5.3 Experiment of implementing a DDoS agent peer. Peer A is a “bad” peer, which may send out a large volume of queries continuously; Peer B is a normal “good” peer, which forwards out as many of the received queries as it is capable of; Peer C is our query traffic observer, which counts all the queries forwarded by B, and does not issue or forward any query**

To better study the characteristics of a DDoS agent, a LimeWire client is modified and an experiment is conducted as illustrated in Figure 5.3. We modified the command line version of LimeWire 2.0.2 by adding a new querying thread to the original source code in peer A. The querying thread reads queries from the log file collected by the monitoring node and issues these queries, and forwards queries coming from the Gnutella network to its neighbors based on the pre-configured time interval. Thus, peer A may work like an overlay DDoS agent, a “bad” peer. Peer B is a “good” peer who never issues any queries and only attempts to forward the queries coming from A. Peer C is a query traffic observer who counts the number of queries forwarded by peer B. Peer C never issues or forwards any queries. The three peers’ network bandwidths are 10M. The machines are all PCs of Dell Optiplex GX300 with P3 733 MHZ CPU and 256M memory.

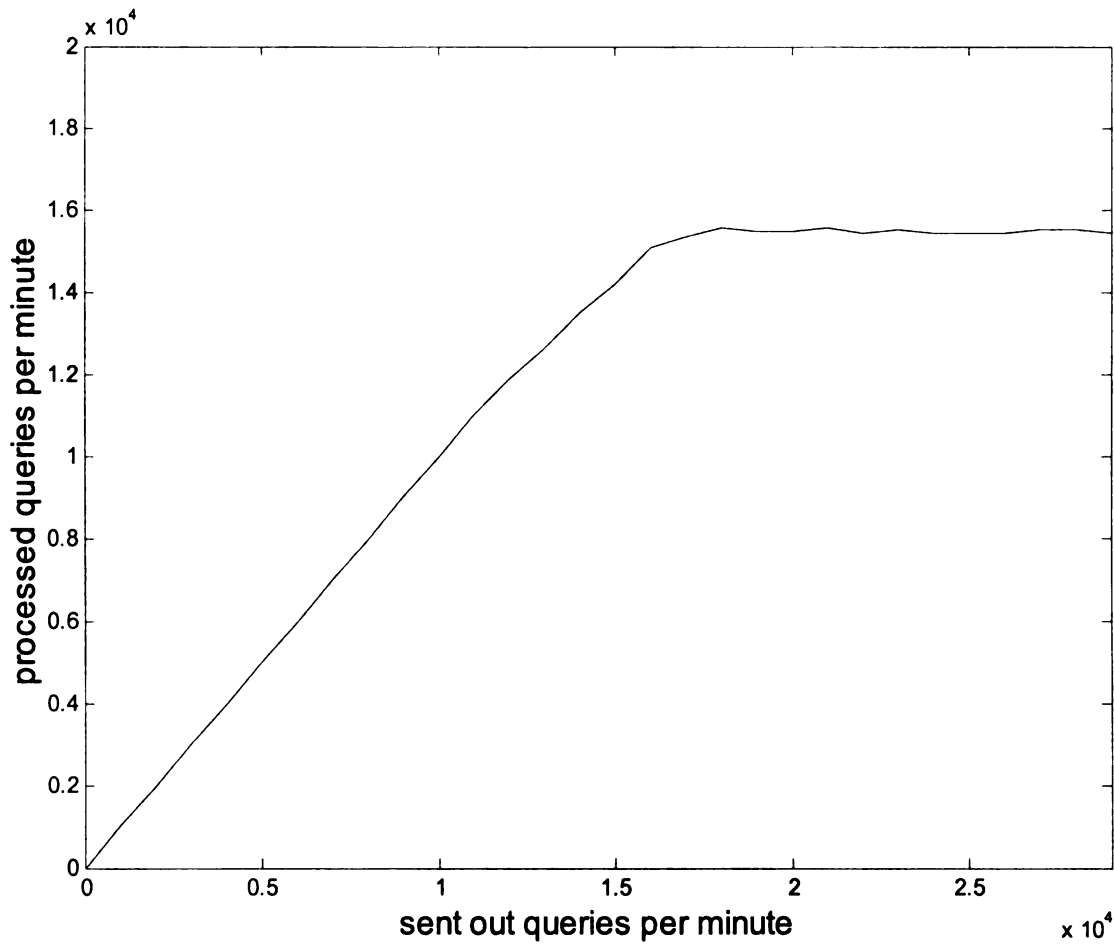
In the experiment, peer A keeps sending queries to peer B at a speed ranging from 1,000 queries per minute to as fast as it could. Indeed, eventually we find Peer A is capable of reading the log file and sending out queries to peer B at a rate of around 29,000 messages per minute. According to Gnutella protocol, for each received query, peer B

will first look up its local sharing storage index, and then forward the query to peer C. Peer C counts the number of queries forwarded by peer B to C so as to measure the capability of a peer in processing search queries and the impact of query flood based DoS on a single Gnutella peer.

We present the experimental results in Figure 5.4. When the number of queries sent out from peer A to B is approaching 15,000 per minute, peer B started discarding queries. In fact, if peer A sends queries to B as fast as it is capable of, 47% of the queries are dropped by peer B. Note that in our experiment both A and B are dedicated to this experiment, while in a real system a normal peer may have other conventional tasks. Furthermore, normally a peer's local index includes many contents; while in our experiment the local index is almost empty, which will reduce time for local look up. Based on these observations, we assume on average a "bad" peer is capable of sending 20,000 queries per minute, and a "good" peer is capable of processing 10,000 queries per minute in the rest of the paper.

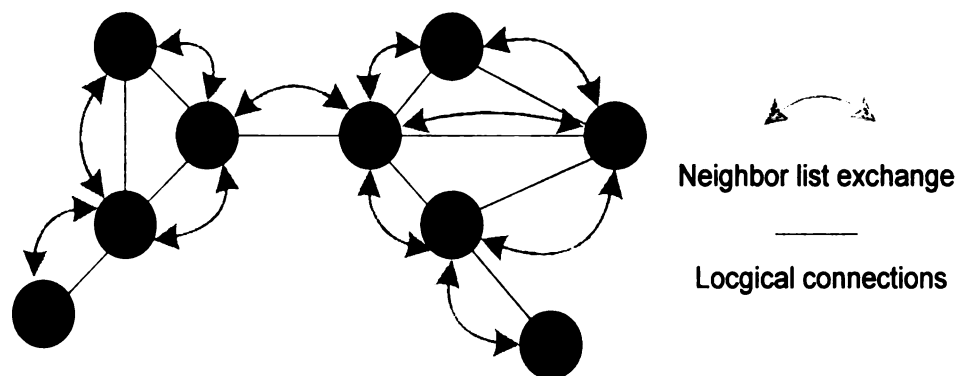
### **5.3 Design of DD-POLICE**

The basic idea of DD-POLICE is that all peers are involved in policing their direct neighbors' query behavior by cooperating with each neighbor's  $r$ -hop away neighbors, and identify the possible "bad" peers for disconnection. For simplicity of the discussion, we first introduce DD-POLICE- $r$  with  $r=1$ , in which a peer will collaborate with all target peer's direct neighbors to identify "bad" peers. Later we will discuss why it is necessary to employ DD-POLICE- $r$  with  $r>1$ . By default, our discussion will focus on DD-POLICE-1.



**Figure 5.4 Queries sent out vs. processed**

DD-POLICE will be employed in decentralized unstructured P2P systems in a fully distributed fashion. It consists of three steps: neighbor list exchanging, neighbor query traffic monitoring, and bad peer recognizing.



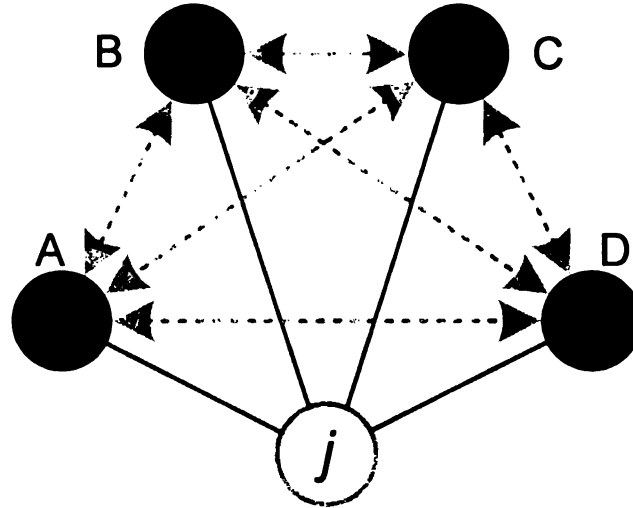
**Figure 5.5 Neighbor List Exchanging**

### 5.3.1 Neighbor List Exchanging

In the design of DD-POLICE, each peer maintains a neighbor list including all its logical neighboring peers. Two neighboring peers exchange their neighbor lists periodically, as illustrated in Figure 5.5.

One issue here is the frequency at which peers exchange their neighbor lists. P2P networks are highly dynamic with peers' joining and leaving randomly. The measurement in [72] indicated that the median up-time for a peer in Gnutella and Napster is 60 minutes. If we assume that a peer's average lifetime is 60 minutes and the exchange frequency is once every 2 minutes, roughly speaking, the probability we miss one or more neighboring peers on detecting query heavy peers in the third step (bad node recognizing) is around 3% ( $2/60$ ). However, simply increase the frequency of exchanging neighbor lists to increase accuracy will cause more overhead to the system. One alternative solution is to make the information exchange event-driven, in which a peer informs all its neighbors whenever its neighboring peer is leaving or a new peer is joining as its neighbor. This solution is favorable to relatively stable networks, but will cause some peers to be very busy during some period of time if the network is highly dynamic. Clearly there is a tradeoff

between overhead and accuracy in the frequency selection. DD-POLICE employs a fixed frequency policy, where a peer sends its neighbor lists to all its neighbors periodically. Our simulation results show that exchanging neighbor list every 2 minutes is a good choice.



**Figure 5.6 An example of a Buddy Group  $BG1-j=\{A,B,C,D\}$**

Another issue deserves some words here is what if a host lies to its neighbor. In our design, when peers exchange their neighbor lists, they will confirm the correctness of the lists with the corresponding peers. A malicious peer could lie about who are its neighbors. If a peer finds out that the claim of a pair of neighboring peers are not consistent, it will disconnect with the one which is its neighbor, and send out a message to both peers indicating the reason of disconnection. In such a way, the good peer in this pair could start to pay more attention to the other peer. If it gets many such messages, the good peer will disconnect with the neighbor.

We define peer  $j$ 's  $r$  hope Buddy Group ( $BGr-j$ ) as the set of peer  $j$ 's neighbors.  $BG1-j$  is as illustrated in Figure 5.6. Depending on how many logical neighbors each peer has, a

peer could belong to multiple different BGs. At the same time, each BG has multiple members. A joining peer creates its BG membership after its first neighbor list exchanging operation. A peer will ping members within the same BG periodically to make sure that other members in the BG are online.

### **5.3.2 Neighbor Query Traffic Monitoring**

Previous studies have shown that queries and desired data have significant locality [76, 89]. A small number of peers issue a large portion of the queries and 5% of files accounts for 50% of all transfers. Peers' behaviors are different in the query frequency and response frequency. Although we do not expect all the peers have exactly the same query pattern, the system expect such fairness: each peer contributes some computation, document, and bandwidth resources, and consumes some search and download services. In other words, any peer issuing extremely huge number of queries is not allowed.

In the step of Neighbor Query Traffic Monitoring, two lists are designed in a peer for each of its logical neighbors, `Out_query(i)` and `In_query(i)`, to record the number of queries per minute from and to the neighboring peer `i`. In our implemented prototype of DD-POLICE-1, a client updates `Out_query(i)` and `In_query(i)` every 2 seconds. At any passing minute, if a neighbor's `Out_query` is greater than a pre-defined threshold, it will be marked as a suspicious DDoS compromised peer.

### **5.3.3 Bad peer recognition**

Based on Gnutella 0.6 protocol, we design a new message type called `Neighbor_Traffic`, which has a message body as shown in Table 3.

**Table 3: Neighbor Traffic message body**

	Source IP Address		Suspect IP Address		Source timestamp		# of Outgoing queries	# of Incoming queries
Byte offset	0	3	4	6	7	9	10	11

When a peer identifies one of its neighbors as a suspicious peer because the neighbor sends out a large amount of queries, this peer will start working with other members in the neighbor's Buddy Group by sending Neighbor\_Traffic messages to them. A Neighbor\_Traffic message includes five fields: Source IP Address, Suspect IP Address, Source timestamp, # of Outgoing queries, # of Incoming queries. The first three fields contain the source IP address of the current peer, the IP address of the suspicious neighbor, and the time the source sends out the message. The last two fields are the number of queries sent out from the source peer to the suspicious peer, and the number of queries that came from the suspicious peer to the source in the past one minute, i.e. Out\_query(suspicious peer) and In\_query(suspicious peer). The payload type of this message can be defined as 0x83. On receiving a Neighbor\_Traffic message, a peer in the BG will check whether it has sent a Neighbor\_Traffic message to other members in this BG in past 5 seconds. If not, it will send such a message to other members.

Let us look at the example in Figure 5.6. Suppose we define the warning threshold as 500 queries per minute, meaning if peer j sends more than 500 queries to peer A in the past minute, A will mark peer j as a suspicious peer. For example, at time t, peer A marks peer j as a 'suspect' and then sends a Neighbor\_Traffic message to each of other members in BG1-j (B, C and D). Since some of members in BG1-j (A, B, C and D) may find peer j suspicious at the same time period, and start to send Neighbor\_Traffic messages to other members, peer A needs to check whether peer B, C or D has sent a

Neighbor\_Traffic message in past 5 seconds. On receiving all the Neighbor\_Traffic messages from B, C and D, or waiting for another 5 seconds, peer A starts to calculate the General Indicator  $g(j, t)$  and the Single Indicator  $s(j, t, A)$ . If any of the two indicators is greater than a predefined value, peer A will affirm peer j as a DDoS compromised peer and disconnect with it.

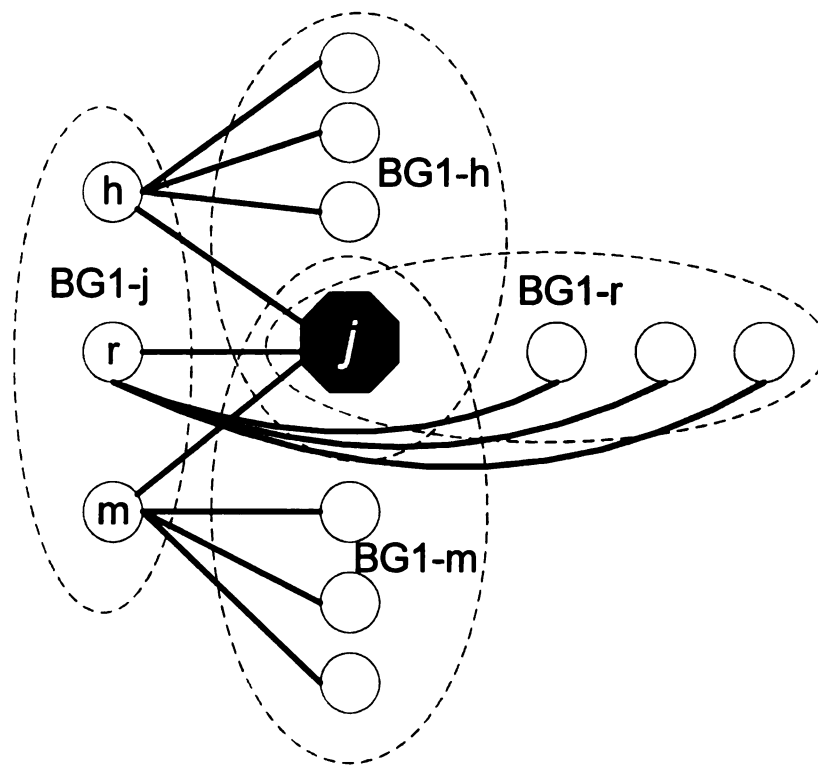
Two key issues should be examined in this step. First, how valid are these two indicators? We have discussed that ideally both of them should reflect  $q_0$ , the number of queries issued by the suspicious peer j. However, it is very hard to expect the General Indicator  $g(j, t)$  and Single Indicator  $s(j, t, A)$  to accurately reflect  $q_0$ . There are three reasons for this: (1) peers are not synchronized, so the data collected by BG members cannot exactly be in the same time period, (2) each peer may receive some duplicated query messages such that the number of incoming queries could be larger than the number of forwarding queries, and (3) peers are frequently coming and leaving, and their average lifetime could be as short as 30 minutes in some systems. As a result, the peer who is calculating  $g(j, t)$  and  $s(j, t, i)$  may fail to receive every BG member's Neighbor\_Traffic message. Nonetheless, our simulation studies and Gnutella DD-POLICE enabled prototype experiments show that the accuracy of  $g(j, t)$  and  $s(j, t, i)$  in representing  $q_0$  is good enough to be used in DD-POLICE, and the misrecognition rate is acceptable low.

The second issue is that of what criteria can convince peer A that peer j is a DDoS compromised peer with very high probability. One choice is to simply make use of definition 5.3, which implies that if one of  $g(j, t)$  and  $s(j, t, i)$  is greater than a threshold, s (in theory, s could be 1), peer A can classify the suspicious peer as a “bad” peer and disconnect with it. However, in real systems, we must carefully choose the threshold. The trade-

off is that the damage of the Overlay DDoS on the system will be controlled by DD-POLICE for low threshold, but some “good” peers may be miscounted as “bad” peers, while a high threshold will lead to “good” peers spending a long time before disconnecting with “bad” peers.

### 5.3.4 An example of DD-POLICE

Let us look at the example shown in Figure 5.7 to see how DD-POLICE works.



**Figure 5.7 An example of DD-POLICE**

In Figure 5.7, peer  $j$  has three neighbors,  $h$ ,  $r$ , and  $m$ , who form a Buddy Group, BG1- $j$ . Meanwhile, peer  $j$  is also involved in three Buddy Groups, which are BG1- $h$ , BG1- $r$  and BG1- $m$ . Suppose peer  $j$  is a DDoS compromised peer, and starts issuing a large amount of queries at time  $t$ . When any of peer  $j$ 's neighbors, such as  $h$ , realizes that peer  $j$  has sent out too many queries, it will exchange query volume information (Neighbor\_Traffic mes-

sages) with the members in BG1-j. If peer r and m are “good” peers, they will inform peer h that they did not send a large amount of queries to peer j. Thus h can figure out that a large amount of queries have been issued by peer j, and may disconnect with peer j immediately.

The question is if suspicious peer j reports a correct number of queries that it sent out to others. Since peer j issues a large amount of queries, one or all of its neighbors may forward a large amount of queries out too. Thus, they could also be questioned by their neighbors. For example, peer m could be suspected by members of BG1-m because peer m is forwarding a large amount of queries for peer j. Peer j belongs to BG1-m. There are three choices for peer j in delivering Neighbor\_Traffic messages to other members in BG1-m: (1) to cheat, (2) not to cheat, or (3) refuse to report. If peer j does not cheat, members in BG1-m will figure out that m is a “good” peer since the majority of outgoing queries from m are the forwarded queries from j instead of initial queries issued by m. We know m will work within BG1-j and disconnect from peer j. There are two cases if peer j chooses to cheat.

*Case 1*, peer j reports a larger number than the number of queries it really sent to peer m. In this case, members in BG1-m will have higher intention to treat peer m as a “good” peer. Peer m will disconnect from peer j without any confusion, and so do the other two members in BG1-j. Thus, this is definitely not a meaningful cheating for peer j.

*Case 2*, peer j reports a smaller number than that the number of queries it really sent to peer m. For example, peer j sent 5,000 queries to peer m in the past minute, but it reports in BG1-m that it has only sent to m 100 queries. As a result, peer m could be treated as a “bad” peer and be disconnected by the other neighbors in BG1-m. However, this choice

has no benefit to peer  $j$  either. On the one hand, making peer  $m$  be wrongly disconnected as a “bad” peer will lead to peer  $j$ ’s attack queries being blocked, while peer  $j$ ’s goal is to attack the P2P system instead of a single peer  $m$ . If all of peer  $j$ ’s neighbors ( $h$ ,  $r$  and  $m$ ) are disconnected by their neighbors, queries issued by peer  $j$  will be isolated among these three peers only, which eliminates the impact of peer  $j$ ’s attack and is not what peer  $j$  wants to achieve. On the other hand, in DD-POLICE, this cheating will not mislead peer  $m$ ’s decision making. Peer  $m$  will work with members in BG1- $j$ , and is able to identify peer  $j$  as the “bad” peer without being confused by peer  $j$ ’s cheating.

The third choice for peer  $j$  is not to report the number of queries it sent to  $m$ , which is the same situation as in Case 2 since, in the design of DD-POLICE, if a peer has not received a Neighbor\_Traffic message from peer  $j$  within a predefined time period, it just assumes that peer  $j$  sent 0 query to peer  $m$ .

In other words, cheating or not reporting will do nothing good for peer  $j$ , and could only degrade the effects of its attacks. Therefore, we assume that peer  $j$  will not cheat in delivering the Neighbor\_Traffic messages.

### **5.3.5 DD-POLICE-r**

As we have discussed, the overlay DDoS attack can be launched by hundreds of peers simultaneously. If each agent works independently, then DD-POLICE-1 is enough to handle. However, two or more connected compromised agents will give DD-POLICE-1 great challenges to identify “bad” peers, especially when all of them lie on the exchanged neighbor list and traffic volume. Therefore, our BG concept is not limited by one-hop neighbors only. Instead, in DD-POLICE-r, each peer’s  $r$ -hop away neighbors are organized to exchange information and to defend against DDoS together. Consequently, peers

working with BGr when  $r > 1$  will incur more traffic overhead, but will make DD-POLICE more powerful. Fortunately, it is not trivial for a DDoS malicious peer to exactly know who are the recruited agents. Thus, making multiple malicious peers connected is much more complicated and difficult than launching DDoS itself. However, we still aim to make the DD-POLICE more powerful and effective into dealing with any possible difficult situations brought by attackers.

Indeed, when peers working with more BG members, more operations are proposed in our research. For example, peers may vote for disconnection with a suspicious peer. They may also share some existing information. That is, if a peer identifies that a specific peer is a “bad” peer, it can send this information to warn other BG members. Since the main purpose of this paper is to propose the concept of Buddy Group and introduce DD-POLICE to defend P2Ps, we will not go through too many details on the designs for defending collaborated malicious peers. In our simulations, we also assume that only few of compromised peers have the ability to work together.

#### **5.4 Performance Metrics**

To evaluate the effectiveness of DD-POLICE on defending overlay DDoS attacks in P2Ps, in addition to traffic cost, query success rate and query response time, we define damage rate and damage recovery time as our main performance metrics as well.

*Damage rate* reflects the service degradation degree when a P2P is under overlay DDoS attacks. In the following definition,  $S(t)$  denotes query success rate of the P2P system when there does not exist any DDoS compromised peers, and  $S'(t)$  denotes the query success rate when the system is under DDoS attack.

**Definition 5.1** Damage rate,  $D(t)$ , is given by:

$$D(t) = \frac{S(t) - S'(t)}{S(t)} \times 100\%$$

*Damage recovery time* is used to evaluate the effectiveness of DD-POLICE in dynamic P2P environment. We define damage recovery time as the time period from when the system damage rate  $D(t)$  is equal or greater than 20% until when the damage is equal or less than 15%. If the damage recovery time is short, this means the DD-POLICE is effective.

We model a malicious node such that it generates as many queries as it is capable of [30]. Measured by our implemented DDoS agent prototype, a “bad” peer could generate more than 20,000 distinct queries per minute. We assign bandwidth to each link based on the observations in [72], which show that 78% of the participating peers have downstream bottleneck bandwidths of at least 100Kbps, and 22% of the participating peers have upstream bottleneck bandwidths of 100Kbps or less. In our simulation, the number of attack queries sent by a compromised peer per minute,  $Q_d$ , is given by:  $Q_d = \min\{20,000, \text{the capacity of the link}\}$ .

## 5.5 Simulation Results

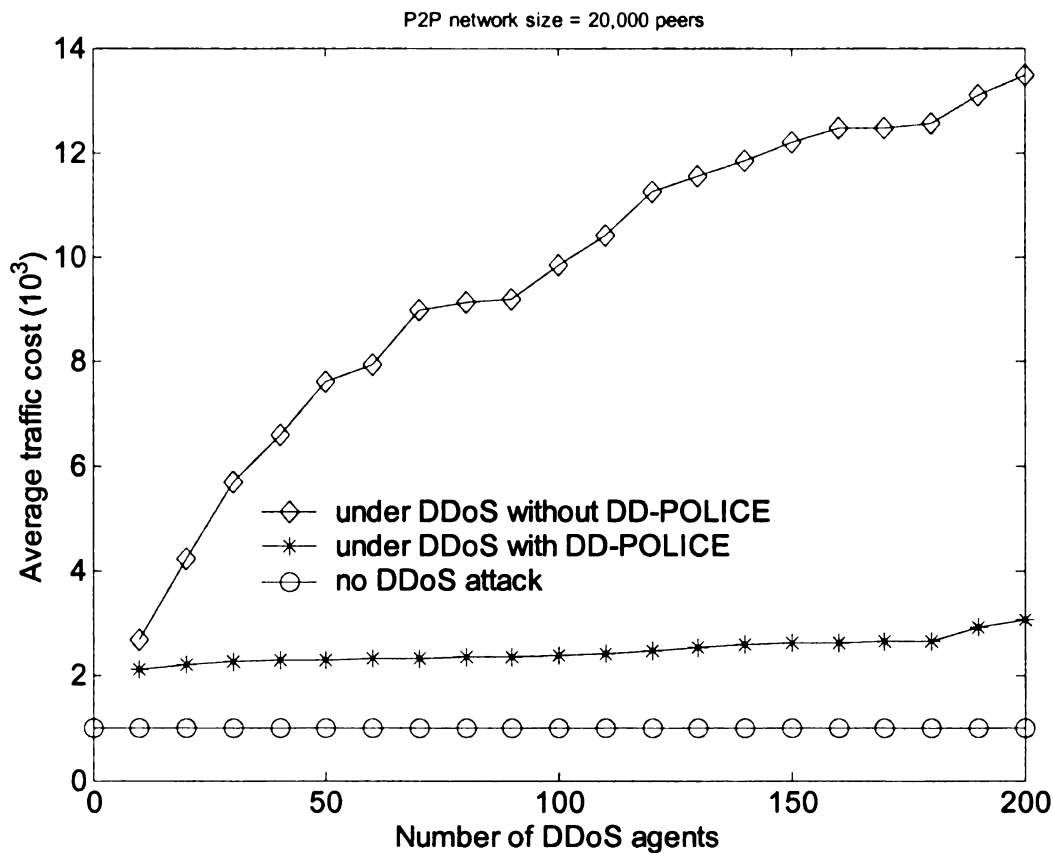
We present our simulation results in this section. Our simulation results on different overlay networks of 20,000 peers on top of 50,000-node Internet-like physical networks are consistent. We representatively show one set of the results.

### 5.5.1 Consequences of overlay DDoS attack in P2Ps

We first quantitatively evaluate how serious the consequences of overlay DDoS attacks are on a P2P network. We generate 1,000,000 queries and track the delivery and response of each query message under a flooding based search mechanism. In each of the simula-

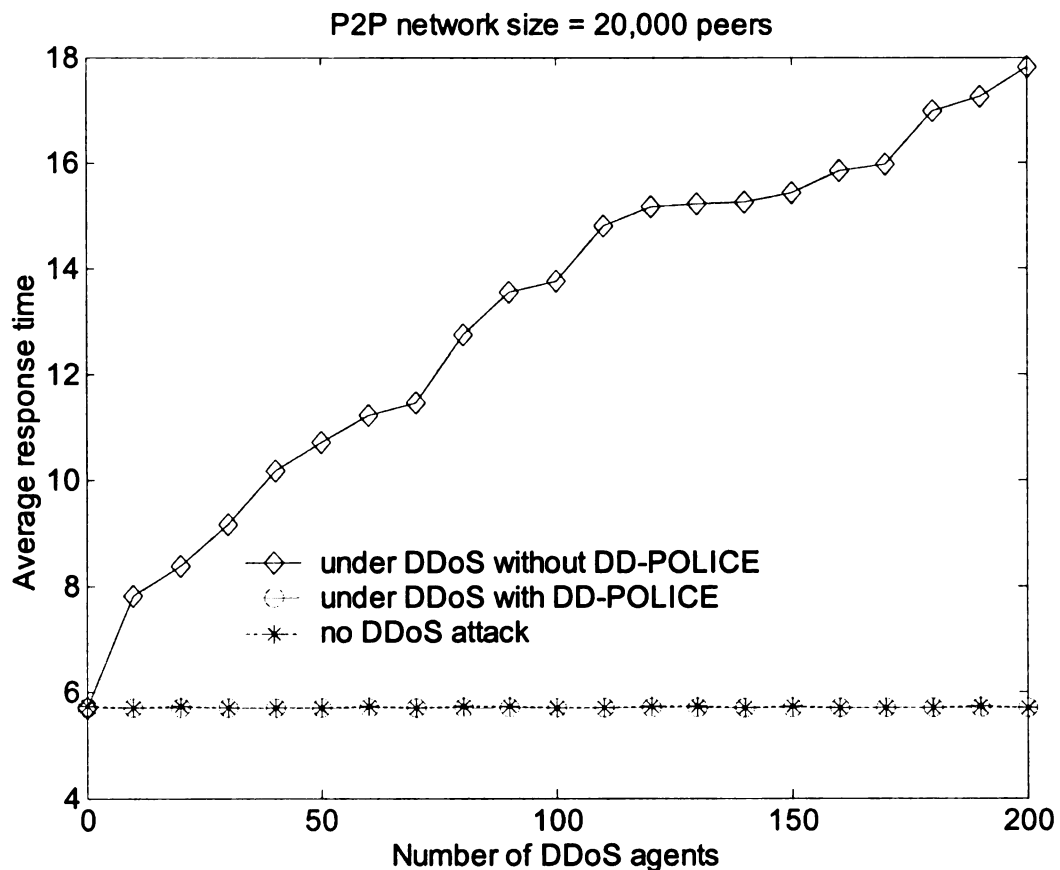
tions,  $k$  random peers, where  $k$  is ranging from 1 to 200, are selected as DDoS compromised peers and each of them keeps sending out attack queries at the maximum rate they are capable of. Meanwhile, normal peers issue queries too. We study the impact of the overlay DDoS attack on a dynamic P2P environment.

Figure 5.8 shows the effect on network traffic. We see that ten to twenty ( $<0.1\%$ ) compromised peers will double the total traffic. When there exists around 100 compromised peers, representing only  $0.5\%$  of the network, the total search traffic increases to 10 times that of the original traffic. The huge amount of search traffic is already a main limit on the scalability of existing popular P2P systems.

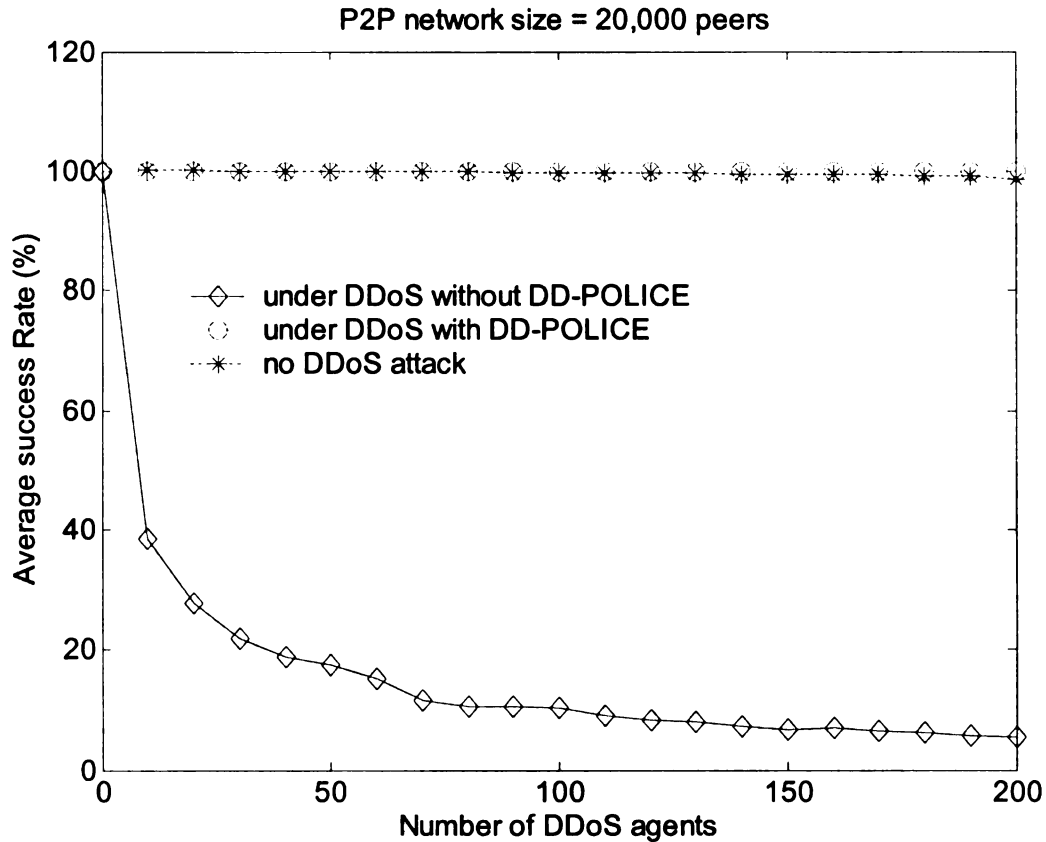


**Figure 5.8 Average traffic cost**

Figure 5.9 and 5.10 show that the service quality of the system is also greatly degraded by DDoS attacks in terms of response time and normalized success rate. We can see that 100 comprised peers will increase the average response time of queries by 2.4 times, and up to 89.7% of queries could fail to receive query responses. These results indicate that, in a real-world P2P system that usually has about 2 millions peers online at any time, less than one thousand DDoS compromised peers could stress the system greatly while ten thousand overlay DDoS agents could overwhelm the whole system.



**Figure 5.9 Query response time**



**Figure 5.10 Success rate**

### 5.5.2 Effectiveness of DD-POLICE

We have discussed that the frequency of neighbor list exchanging is one of the key issues to be examined. We have simulated and evaluated two policies: (1) periodic policy: to exchange neighbor lists every  $s$  minutes, where  $s$  ranges from 1 to 10; and (2) event driven policy: a peer reports its neighbor list whenever a new neighbor is joining or an existing neighbor is leaving.

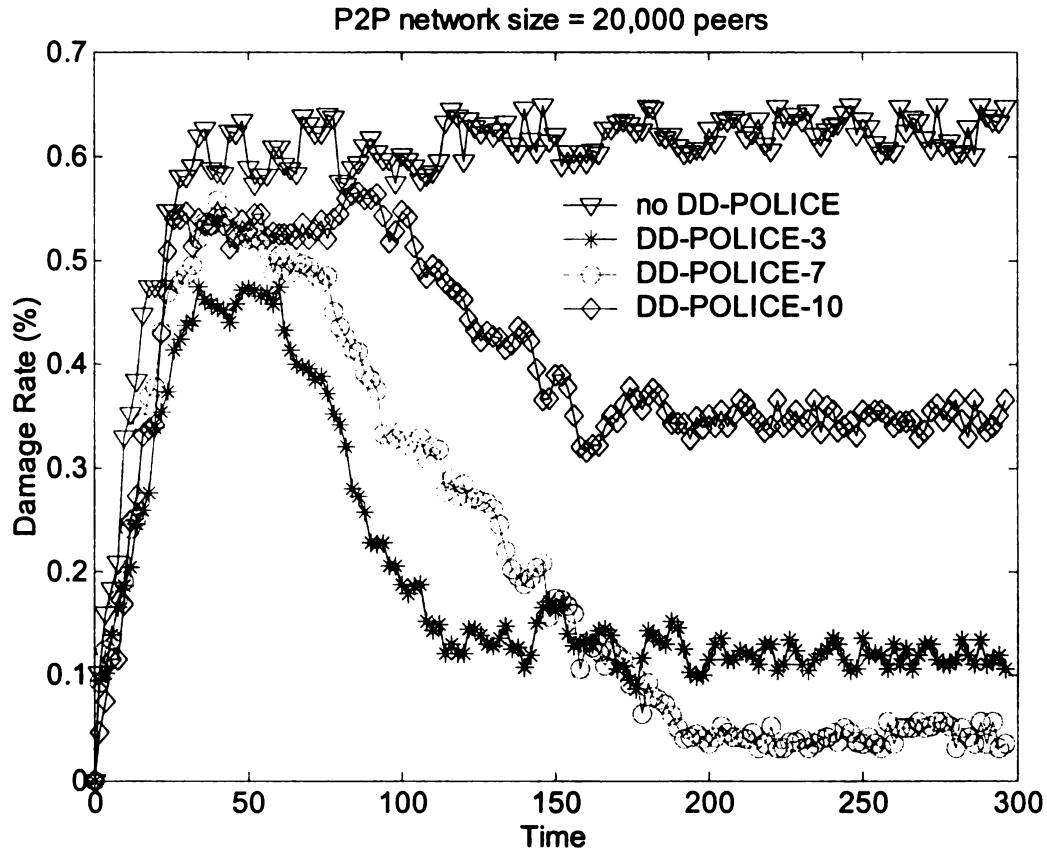
As we observed from our simulation results, there is no big difference on the overall performance by using the first policy as long as  $s$  is no more than 2 minutes, while the second police incurs much higher traffic overhead compared with the first policy because

the P2P network is very dynamic. But if we further increase  $s$ , such as to 4 or 5 minutes, the possibility of DD-POLICE misjudging “good” peer and “bad” peers will be greatly increased because of the inaccurate neighbor lists. Therefore, in other simulations and the DD\_POLICE implementation prototype, we use periodic policy in which peers report their neighbor lists every 2 minutes.

Another key issue to examine is when a peer should make the decision to disconnect a suspicious neighbor as a “bad” peer.

The decision is based on two calculated indicators  $g(j, t)$  and  $s(j, t, i)$ . In DD-POLICE, we define CT as a cut threshold. If the value of  $g(j, t)$  or  $s(j, t, i)$  computed by peer  $i$  is greater than CT, peer  $i$  will disconnect from peer  $j$ . CT is 1 in definition 5.3. However, we have mentioned that  $g(j, t)$  and  $s(j, t, i)$  are based on some assumptions which may affect the accuracy of the estimation, and the dynamic changing of the overlay topology also makes the choice of CT a challenging endeavor. The dilemma is that a smaller value of CT may mislead the peers to wrongly disconnect some “good” peers, while a larger value of CT could allow some “bad” peers to avoid disconnection. In order to select an optimal value of CT, we study the impact of different values of CT using three kinds of errors: false negative is the number of “good” peers that are wrongly disconnected, false positive is the number of “bad” peers that are not identified and not disconnected, and false judgment is the sum of the above two.

Another consideration in the choice of CT is the convergent speed of the algorithm. Decentralized P2Ps are fully distributed systems. When some peers recognize a “bad” peer, the only thing these peers can do is to disconnect with the “bad” peer. No mechanism can prevent the DDoS Agent from joining the system



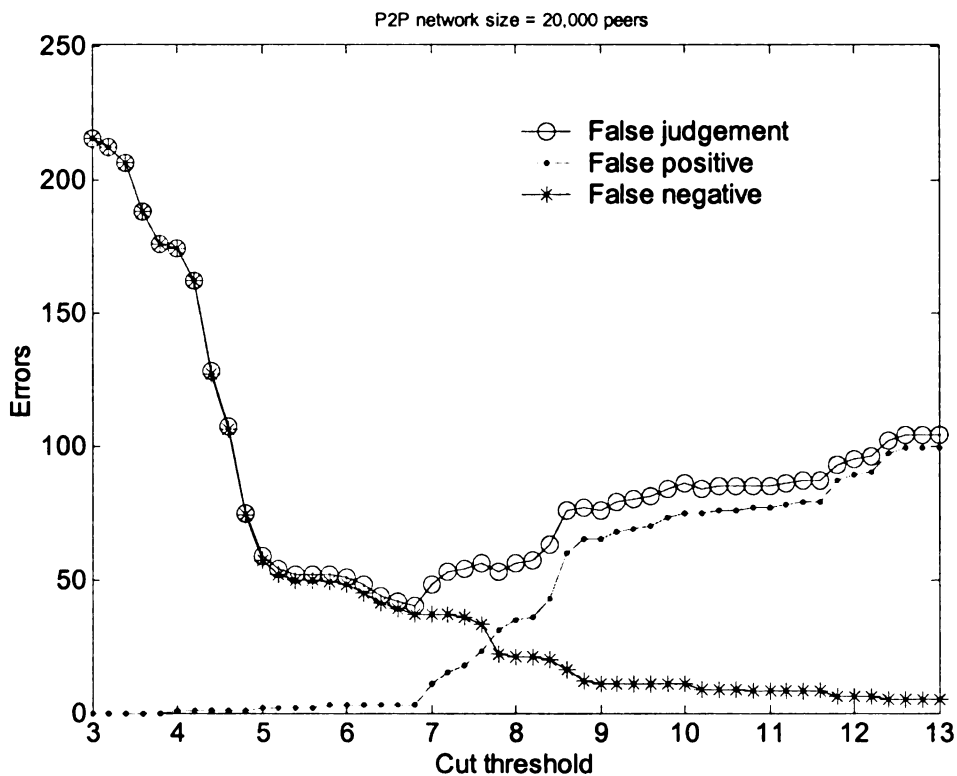
**Figure 5.11 Effectiveness of DD-POLICE in Dynamic P2P environments**

again and launching another round of attacks. We desire our approach to identify “bad” peers in a very short period of time in a dynamic P2P environment. The metric of damage recovery time is used to evaluate the performance of DD-POLICE for this consideration. We strive for short damage recovery time.

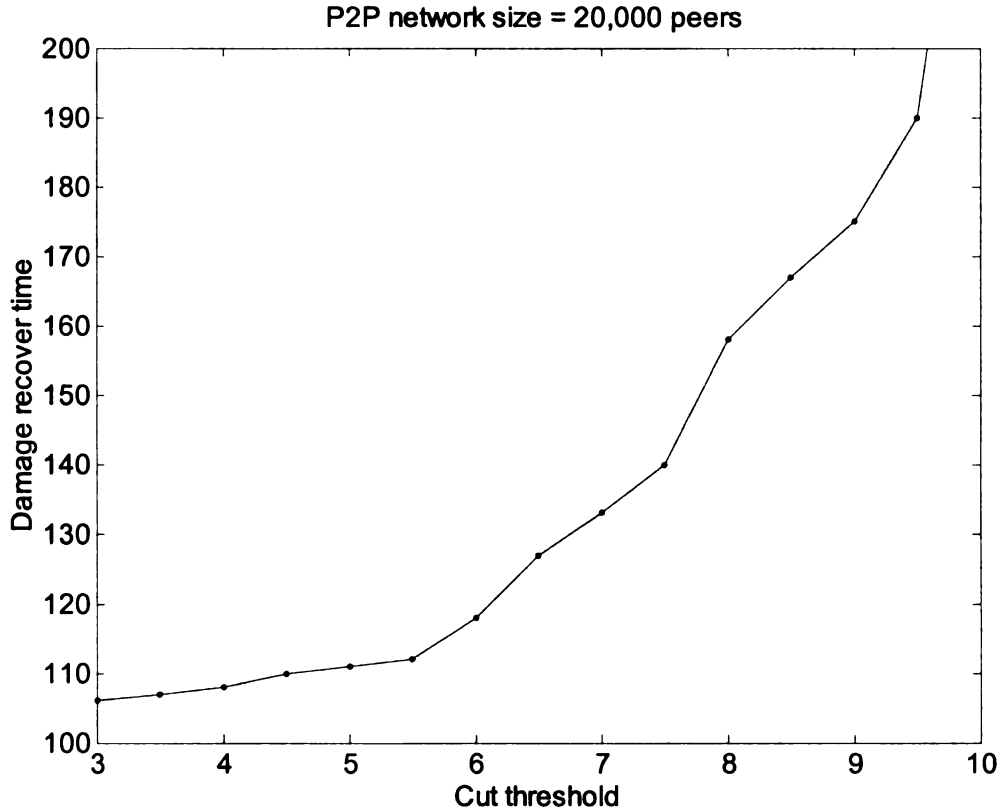
Figures 5.11-5.13 show the impact of CT on the performance of DD-POLICE when there are 100 DDoS agents in a 20,000-peer system. DD-POLICE-n means DD-POLICE scheme with  $CT=n$ . Figure 5.11 shows that DD-POLICE with  $CT=3$  reduces the damage of attack faster than DD-POLICE with  $CT=7$ . However, the damage rate of  $CT=3$  scheme cannot be reduced as low as that of  $CT=7$  scheme. The reason is that, compared with  $CT=7$  scheme, more “good” peers are misjudged as “bad” peers and disconnected by

their neighbors in CT=3 scheme, causing a lower success rate. However, CT cannot be too large. Figure 5.11 shows that with CT=10, DD-POLICE converges very slowly and the stabilized damage rate is much higher than that in CT=3 and CT=7.

We further show the three kinds of errors in Figure 5.12 and damage recovery time in Figure 5.13. We can see that in Figure 5.12, as CT increases, the false negative decreases, while the false positive increases. That means when we set a larger cut threshold, fewer “good” peers will be misjudged as “bad” peers and more “bad” peers will be misjudged as “good” peers. The false judgment is optimal when CT is within 5 to 7 in Figure 5.12. Figure 5.13 shows that as CT increases, DD-POLICE needs a longer time to identify a peer as a “bad” one. Thus, the damage recovery time is longer. Comprehensively considering the performance of DD-POLICE, we choose CT=5 or 6.



**Figure 5.12 Errors vs. cut threshold**



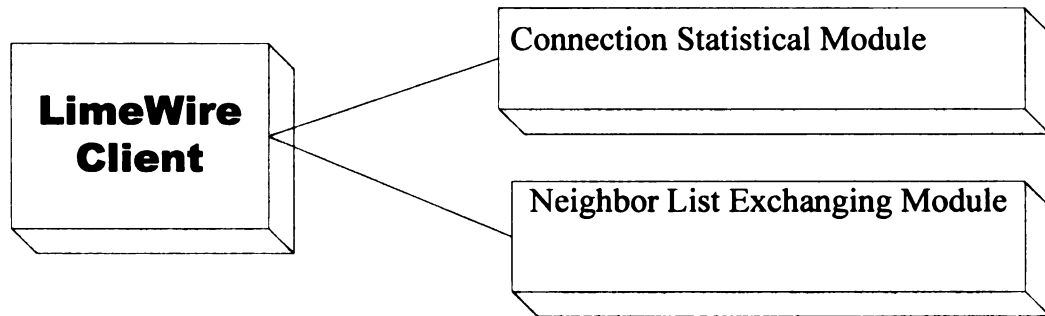
**Figure 5.13 Damage recovery time vs. cut threshold**

### 5.5.3 DD-POLICE in Dynamic P2P Environments

Adding DD-POLICE into the peers in the dynamic simulation environment, we repeat the described experiment with the same 1,000,000 queries. Basic setup of DD-POLICE includes exchanging neighbor lists every 2 minutes and  $CT=5$ . Figures 5.8-5.10 show that DD-POLICE effectively reduces the damage of overlay DDoS and is very scalable. Compared with the case of “no DDoS attack”, DD-POLICE achieves a comparable average response time and success rate with slightly higher average traffic cost.

## 5.6 Implementation of DD-POLICE

To further investigate the feasibility of DD-POLICE in practical P2P systems, we implement a prototype of DD-POLICE based on Gnutella protocol v0.6. Figure 5.14 illustrates our basic modification on a LimeWire client with Gnutella protocol v0.6 to implement DD-POLICE. The Connection Statistical

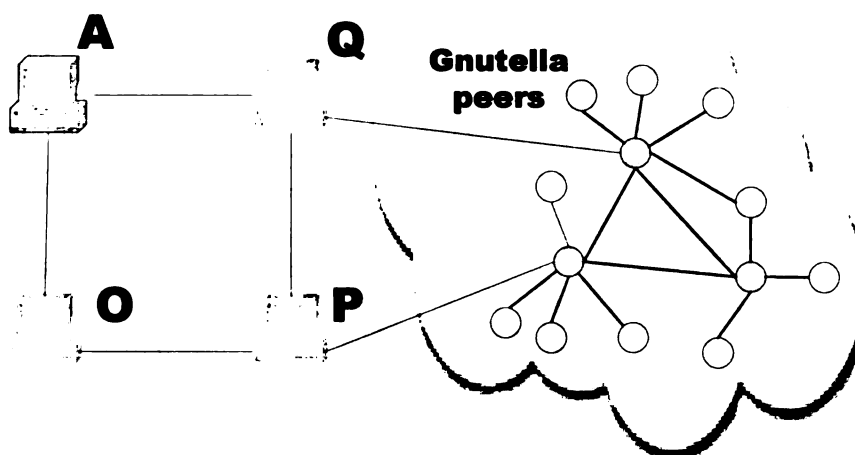


**Figure 5.14 Prototype of a DD-POLICE enabled client**

Module records the number of queries sent to the neighbors through the connections, and the number of queries received from the neighbors through the same connections. The Neighbor List Exchanging Module keeps exchanging the neighbor lists with neighboring peers every two minutes so that each node has a list of its neighbors' neighbors.

When a DD-POLICE enabled client finds that a suspicious neighbor is sending queries with very high rate, it will start working with all the neighbors of the suspicious peer. It picks up IP addresses from the neighbor list of the suspicious peer, and sets up temporary connections with these neighbors. Through the temporary connections, Neighbor\_Traffic messages will be exchanged. In order to accurately collect the number of queries sent to

the suspicious node from those neighbors, the temporary connections between DD-POLICE enabled clients in our prototype have higher priority than other connections.



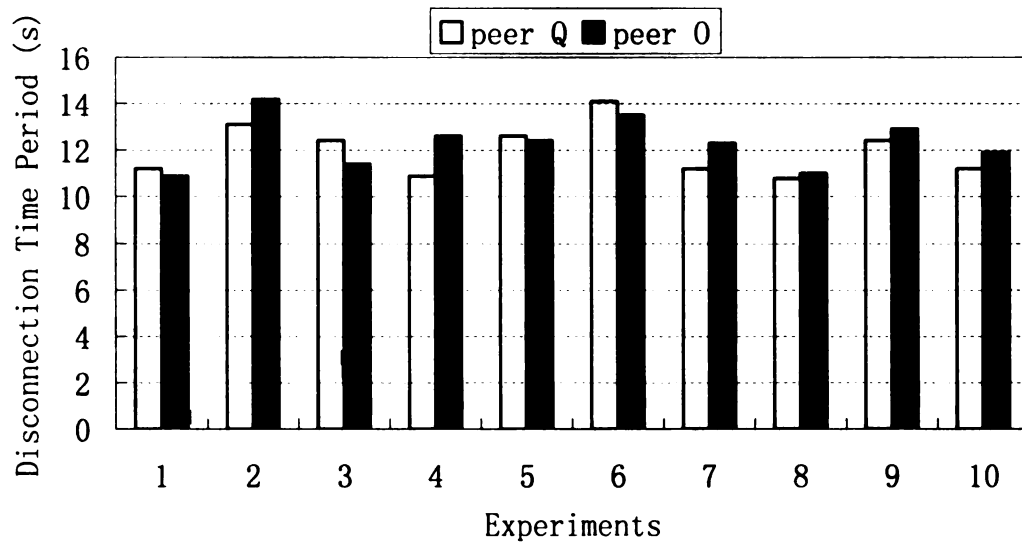
**Figure 5.15 A simple experiment of DD-POLICE**

We then conduct our experiments in a setup shown in Figure 5.15, where peer A is the “bad” peer who keeps sending out queries to peers Q and O. Peers O, P Q are all DD-POLICE enabled peers. The basic setups of DD-POLICE include that the timeout for Query\_traffic message collection is 5 seconds, cut threshold is  $CT=5$ , neighbor lists exchange is every 2 minutes, and a peer is defined as a suspicious peer on receiving more than 500 queries in the past minute.

At the beginning of each experiment, peer A works as a “good” peer. It issues less than tens of queries per minute on average. After a certain period of time, peer A randomly picks up 30,000 queries from the query trace we collected, and starts sending the queries out at a rate of 6,000 per minute to each of peer O and peer Q. We repeat the same experiment 10 times. Each time, peers O and Q can successfully disconnect with peer A

within less than 15 seconds from when peer A starts ‘launching’ the attack, as shown in Figure 5.16.

Although the experimental setup is relatively simple, we can see that DD-POLICE is easy to implement and is effective in dealing with overlay DDoS attacks. Widely deploying DD-POLICE will help P2P systems defend against overlay DDoS attacks.



**Figure 5.16 Experimental Results of DD-POLICE Implementation. Both peer Q and peer O may successfully disconnect with the malicious peer A within 15 seconds from when peer A starts flooding a large volume of queries to them**

## 5.7 Discussion

Unstructured P2P systems are most commonly used in practice, but are vulnerable to overlay DDoS attacks. Most previous techniques protect networks from network-layer DDoS attacks and cannot be applied to overlay DDoS attacks. Overlay flooding-based DDoS attacks can be more damaging in that a small number of messages are inherently propagated to consume a large amount of bandwidth and computation resources.

In this chapter, we propose a distributed and scalable method, DD-POLICE, to detect malicious nodes in order to defend P2P systems from the DDoS attacks. Our study of DD-POLICE is admittedly in its early stages and there are several issues that need to be discussed and studied in the future. For example, we need to explore a mechanism in DD-POLICE to share previous information. P2Ps are fully distributed systems, and our DD-POLICE only helps individual peers identify “bad” peers while the other peers do not learn this information. The “bad” peers may create connections with others and launch attacks. However, to share this information in P2P is extremely challenging because there is no dedicated central server, and it is hard to know whether a peer is telling the truth or not. Further, most agents are not the DDoS attackers, and the system should not block them forever. Even if all these problems are solved, to dynamically distribute the information is not trivial in that large amount of traffic will be incurred.

Another issue is the working mode of DD-POLICE- $r$ . Clearly DD-POLICE-1 is simple but only works against isolated malicious hosts. Employing DD-POLICE- $r$  with  $r > 1$  means more overhead. We are going to explore a mechanism to determine the scope of using DD-POLICE in different depth of peers’ neighbors.

Other future work includes employing and measuring DD-POLICE in larger scale systems, such as in PlanetLab[66].

## 6 Conclusion and Future Work

The peer-to-peer model has recently gained significant attention due to its high potential of sharing various resources among a large number of networked users, where each peer acts as both a resource provider and a consumer. Unstructured P2P systems are most commonly used in practice, but are not scalable and secured.

### 6.1 *Summary*

An attractive feature of P2P overlay networks is that peers do not need to directly interact with the underlying physical network. This provides many new opportunities for user level development and applications. However, the mechanism for a peer to randomly choose logical neighbors without any knowledge about the physical topology causes a serious topology mismatch between P2P overlay networks and the physical network. Because of the mismatching problem, a pair of logical neighbors can be far away from each other, causing a message to traverse the same physical link multiple times, and wasting huge amount of network bandwidths in the process. Our experimental results show that about 70% of the paths suffer from the topology mismatching problems. The performance gain of any improved file searching schemes is seriously hindered by this topology mismatching problem. The mismatching problem and flooding-based search in unstructured

P2P systems cause heavy network traffic that contributes the largest portion of overall internet traffic.

In this dissertation, we have proposed several approaches to solve overlay topology mismatch problems in P2P systems. They are scalable and completely distributed in the sense that they do not require global knowledge of the whole overlay network when each node is optimizing the organization of its logical neighbors. For example, in Adaptive Overlay Topology Optimization (AOTO), every single peer builds an overlay multicast tree between itself (source node) and the peers within a certain diameter from the source peer, and then optimizes the neighbor connections that are not on the tree, while retaining the search scope. Our simulations show that AOTO can significantly improve the performance of P2P systems.

To improve the convergence speed of AOTO, we propose a location-aware topology matching (LTM) scheme [2]. In LTM, each peer issues a detector in a small region so that the peers receiving the detector can record relative delay information. Based on the delay information, a receiver can detect and cut most of the inefficient and redundant logical links, as well as add closer nodes as its direct neighbors. Our simulation studies show that the total traffic and response time of the queries can be significantly reduced by LTM without shrinking the search scope. Our study shows that only one tenth of the original traffic cost is necessary to cover the same number of peers, and the average response time is reduced by approximately 80%.

To further remove traffic overhead, we propose a scalable bipartite overlay (SBO)[3]. The SBO which employs an efficient strategy for distributing optimization tasks in peers with different colors. In SBO, each joining peer is assigned a color so that all peers are

divided into two groups of white or red colors, respectively. Each peer is only connected with peers in its opposite color. Each white peer probes neighbor distances and reports the information to the red neighbors. Each red peer computes efficient forwarding paths. A white peer that is not on forwarding paths of a red peer then tries to find a more efficient red peer to replace this red neighbor. Our evaluations show that SBO achieves approximately 85% reduction on traffic cost and more than 55% reduction on query response time.

In above mentioned three approaches, LTM has the fastest convergent speed, which is important in a highly dynamic P2P environment. However, LTM needs NTP support, which means extra overhead. To retain the fast convergent speed of LTM without the need of synchronizing peering nodes, we design THANCS. In THANCS, each peer probes the distances with its immediate logical neighbors and piggybacks all its neighbors' distance information with selected query messages. Thus, peers may have two hop away neighbors' information without much extra overhead to optimize the topology to approach an optimal overlay. With THANCS enabled, total network traffic incurred by blind flooding search will be decreased by 75% without shrinking the search scope. At the same time, average query response time is also decreased by approximately 60-65%.

To effectively protect against overlay DDoS attacks, we propose a detection based approach, DD-POLICE. In DD-POLICE, peers monitor the traffic to and from each of their neighbors. If a peer receives a very large number of queries from one of its neighbors, it will mark this neighbor as a suspicious DDoS peer. This peer will then exchange query volume information with the suspicious peer's neighbors so that it can figure out the number of queries originally issued by the suspicious peer. Based on this number, this

peer makes a final decision on whether the suspicious neighbor is a DDoS peer and should be disconnected. Through comprehensive simulations, we demonstrate the serious impact of overlay DDoS attacks on P2P systems, and the effectiveness of DD-POLICE in dynamic P2P environments. We then implement a prototype of a DD-POLICE enabled client, and show that DD-POLICE is easy to use and effective on defending against overlay DDoS in P2P systems.

## **6.2 *Future Work***

Future work will lead into three directions.

Firstly, pervasive computing and sensor networking will inevitably play an important role in the near future. They share some similar distributed natures with P2P systems, such as self-organization, broadcasting, and non-fix infrastructure. It is very natural to connect my current research in P2P systems to pervasive computing and sensor networking. There are many interesting problems in these realms with regards to efficiency, scalability, reliability and security.

Secondly, I will continue my research on optimizing overlay topologies of P2P systems. Existing approaches are all heuristic and we intend to design an offline topology optimization algorithm. It might be very hard to have an optimal topology matching algorithm, but it can be a sub-optimal algorithm so that it can be a benchmark to evaluate all of the proposed topology optimization techniques. In addition, we will evaluate and employ existing orthogonal optimization approaches and combine them into a new practical protocol, then implement it PlanetLab environment.

Finally, we plan to extend current research on network security of P2P systems. Besides defending against overlay DDoS, further protocols under different P2P systems will

be proposed on two critical issues: data integrity and communication anonymity. Data integrity in P2P systems is important in the sense that peers can cache files downloaded from other peers for potential sharing source to some other peers, so that the same files can be provided by multiple peers. Mutual anonymity is defined as being when neither the initiator nor the responder can identify each other, and no other peers can identify the two communication parties with certainty. We intend to design new protocols aimed at achieving mutual anonymity, data integrity and efficiency in P2P systems.

## References

- [1] BRITE, <http://www.cs.bu.edu/brite/>
- [2] CERT Advisory CA-1998-01 Smurf IP Denial-of-Service Attacks, <http://www.cert.org/advisories/CA-1998-01.html>
- [3] Cisco, "Strategies to protect against distributed denial of service attacks," <http://www.cisco.com/warp/public/707/newsflash.html>
- [4] D. Dittrich, "The DoS project's 'Trinoo' distributed denial of service attack tool," Oct. 1999; "The 'Stacheldraht' distributed denial of service attack tool," Dec. 1999; "The 'Tribe Flood Network' distributed denial of service attack tool," Oct. 1999, <http://www.washington.edu/People/dad>
- [5] D. Dittrich, G. Weaver, S. Dietrich, and N. Long. The mstream distributed denial of service attack tool., <http://netsec.gsfc.nasa.gov/~spock/mstream.analysis.txt>
- [6] D. Dittrich, S. Dietrich, N. Long, "An analysis of the 'shaft' distributed denial of service tool," March 2000, [http://netsec.gsfc.nasa.gov/~spock/shaft\\_analysis.txt](http://netsec.gsfc.nasa.gov/~spock/shaft_analysis.txt)
- [7] Fasttrack, <http://www.fasttrack.nu>
- [8] Gnutella, <http://gnutella.wego.com/>
- [9] The Gnutella protocol specification 0.6, <http://rfc-gnutella.sourceforge.net>
- [10] KaZaA, <http://www.kazaa.com>
- [11] Limewire, <http://www.limewire.com>
- [12] Napster, <http://www.napster.com>
- [13] NLANR, National Laboratory for Applied Network Research, <http://moat.nlanr.net/Routing/rawdata>, 2001
- [14] NTP: The Network Time Protocol, <http://www.ntp.org/>
- [15] Tripwire, "Tripwire for servers," <http://www.tripwire.com/products/servers/>
- [16] V. Almeida, A. Bestavros, M. Crovella, and A. d. Olivera, "Characterizing Reference Locality in the WWW," *Proceedings of the IEEE Conference on Parallel and Distributed Information Systems (PDIS)*, 1996.
- [17] R. Bhagwan, S. Savage, and G. M. Voelker, "Understanding Availability," *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems(IPTPS)*, 2003.

- [18] R. Bhagwan, S. Savage, and G. M. Voelker, "Understanding Availability," *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.
- [19] B. Bhattacharjee, S. Chawathe, V. Gopalakrishnan, P. Keleher, and B. Silaghi, "Efficient Peer-to-peer Searches Using Result-caching," *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.
- [20] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," *Proceedings of IEEE INFOCOM*, 1999.
- [21] T. Bu and D. Towsley, "On Distinguishing between Internet power law topology generators," *Proceedings of IEEE INFOCOM*, 2002.
- [22] R. K. C. Chang, "Defending against Flooding-Based Distributed Denial-of-Service Attacks: A Tutorial," *IEEE Communications Magazine*, pages: 42-51, October, 2002.
- [23] G. Chartrand and O. R. Oellermann, "Applied and Algorithmic Graph Theory," *McGraw-Hill, New York*, 1993.
- [24] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-like P2P Systems Scalable," *Proceedings of ACM SIGCOMM*, 2003.
- [25] Y. Chu, S. G. Rao, and H. Zhang, "A Case for End System Multicast," *Proceedings of ACM SIGMETRICS*, 2000.
- [26] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," *Design Privacy Enhancing Technologies, Workshop on Design Issues in Anonymity and Unobservability, LNCS 2009, ed. by H. Federrath. Springer-Verlag, pp. 46-66*, 2001.
- [27] E. Cohen, H. Kaplan, and A. Fiat, "Associative Search in Peer-to-peer Networks: Harnessing Latent Semantics," *Proceedings of IEEE INFOCOM*, 2003.
- [28] E. Cohen and S. Shenker, "Replication Strategies in Unstructured Peer-to-peer Networks," *Proceedings of ACM SIGCOMM*, 2002.
- [29] A. Crespo and H. Garcia-Molina, "Routing Indices for Peer-to-peer Systems," *Proceedings of 28th Conference on Distributed Computing Systems*, 2002.
- [30] N. Daswani and H. Garcia-Molina, "Query-Flood Dos Attacks in Gnutella," *ACM Computer and Communications Security*, 181-192, 2002.
- [31] N. Daswani, H. Garcia-Molina, and B. Yang, "Open Problems in Data-Sharing Peer-to-Peer Systems," *Proceedings of the 9th International Conference on Database Theory (ICDT)*, 2003.

- [32] R. Dingledine, M. J. Freedman, and D. Molnar, "The Free Haven Project: Distributed Anonymous Storage Service," *Workshop on Design Issues in Anonymity and Unobservability, LNCS 2009*, ed. by H. Federrath. Springer-Verlag, pp.67-95, 2001.
- [33] N. Duffield, F. Presti, V. Paxson, and D. Towsley, "Inferring Link Loss Using Striped Unicast Probes," *Proceedings of IEEE INFOCOM*, 2001.
- [34] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing," *RFC 2827*, 2000.
- [35] I. Foster and A. Iamnitchi, "On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing," *Proceedings of 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.
- [36] M. J. Freedman, E. Sit, J. Cates, and R. Morris, "Introducing Tarzan, a peer-to-peer anonymizing network layer," *Proceedings of the 1st International Workshop on Peer-to-peer Systems*, 2002.
- [37] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," *Freeman, New York*, 1979.
- [38] O. D. Gnawali, "A Keyword-Set search system for peer-to-peer networks," in *Master's thesis, Massachusetts Institute of Technology*, June, 2002.
- [39] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload," *Proceedings of 19th ACM Symposium on Operating Systems Principles (SOSP)*, 2003.
- [40] J. Ioannidis and S. M. Bellovin, "Implementing Pushback: Router-Based Defense Against DDoS Attacks," *Proceedings of NDSS*, 2002.
- [41] B. J. Jansen, A. Spink, J. Bateman, and T. Saracevic, "Real Life Information Retrieval: a study of User Queries on the Web," *Proceedings of SIGIR Forum*, 1998.
- [42] S. Jiang, L. Guo, and X. Zhang, "LightFlood: An Efficient Flooding Scheme for File Search in Unstructured Peer-to-Peer Systems," *Proceedings of International Conference on Parallel Processing (ICPP)*, 2003.
- [43] S. Jiang and X. Zhang, "FloodTrail: An Efficient File Search Technique in Unstructured Peer-to-peer Systems," *Proceedings of IEEE GLOBECOM*, 2003.
- [44] S. Joseph, "NeuroGrid: Semantically Routing Queries in Peer-to-Peer Networks," *Proceedings of the International Workshop on Peer-to-Peer Computing*, 2002.
- [45] A. Juels and J. Brainard, "Client puzzles: A cryptographic countermeasure against connection depletion attacks," *Proceedings of Networks and distributed system security symposium (NDSS)*, 1999.

- [46] A. D. Keromytis, V. Misra, and D. Rubenstein, "SOS: Secure Overlay Services," *Proceedings of ACM SIGCOMM*, 2002.
- [47] B. Krishnamurthy and J. Wang, "Topology Modeling via Cluster Graphs," *Proceedings of SIGCOMM Internet Measurement Workshop*, 2001.
- [48] C. Lindemann and O. P. Waldhorst, "A Distributed Search Service for Peer-to-Peer File Sharing in Mobile Applications," *Proceedings of the International Workshop on Peer-to-Peer Computing*, 2002.
- [49] Y. Liu, X. Liu, C. Wang, and L. Xiao, "Defending P2Ps from overlay flooding based DDoS," *submitted for publication*, 2004.
- [50] Y. Liu, X. Liu, L. Xiao, L. M. Ni, and X. Zhang, "Location-Aware Topology Matching in Unstructured P2P Systems," *Proceedings of IEEE INFOCOM*, 2004.
- [51] Y. Liu, L. Xiao, and L. M. Ni, "Building a Scalable Bipartite P2P Overlay Network," *Proceedings of 18th International Parallel and Distributed Processing Symposium (IPDPS)*, 2004.
- [52] Y. Liu, L. Xiao, L. M. Ni, and Y. Liu, "Overlay Topology Matching in Unstructured P2P Systems," *Proceedings of the Second International Workshop on Grid and Cooperative Computing (GCC)*, 2003.
- [53] Y. Liu, Z. Zhuang, L. Xiao, and L. M. Ni, "AOTO: Adaptive Overlay Topology Optimization in Unstructured P2P Systems," *Proceedings of IEEE GLOBECOM*, 2003.
- [54] Y. Liu, Z. Zhuang, L. Xiao, and L. M. Ni, "A Distributed Approach to Solving Overlay Mismatch Problem," *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS)*, 2004.
- [55] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-peer Networks," *Proceedings of the 16th ACM International Conference on Supercomputing*, 2002.
- [56] E. P. Markatos, "Tracing A Large-scale Peer-to-peer System: An Hour in The Life of Gnutella," *Proceedings of the 2nd IEEE/ACM International Symp. on Cluster Computing and the Grid*, 2002.
- [57] D. A. Menasce and L. Kanchanapalli, "Probabilistic Scalable P2P Resource Location Services," *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, pp. 48-58, 2002.
- [58] J. Mirkovic, G. Prier, and P. Reiher, "Attacking DDoS at the Source," *Proceedings of ICNP*, 2002.
- [59] J. Mirkovic, G. Prier, and P. Reiher, "Challenges of Source-End DDoS Defense," *Proceedings of NCA*, 2003.

- [60] J. Mirkovic and P. Reiher, "A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms," *UCLA CSD Technical Report no. 020018*.
- [61] D. Moore, G. Voelker, and S. Savage, "Inferring Internet Denial-of-Service Activity," *Proceedings of Usenix Security Symposium*, 2001.
- [62] W. G. Morein, A. Stavrou, D. L. Cook, A. D. Keromytis, V. Misra, and D. Rubenstein, "Using Graphic Turing Tests to Counter Automated DDoS Attacks Against Web Servers," *Proceedings of ACM International Conference on Computer and Communications Security (CCS)*, 2003.
- [63] A. Nakao, L. Peterson, and A. Bavier, "A Routing Underlay for Overlay Networks," *Proceedings of ACM SIGCOMM*, 2003.
- [64] V. N. Padmanabhan and L. Subramanian, "An Investigation of Geographic Mapping Techniques for Internet Hosts," *Proceedings of ACM SIGCOMM*, 2001.
- [65] S. Patro and Y. C. Hu, "Transparent Query Caching in Peer-to-Peer Overlay Networks," *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS)*, 2003.
- [66] L. Peterson, D. Culler, T. Anderson, and T. Roscoe, "A blueprint for introducing disruptive technology into the Internet," *Proceedings of HOTNETS*, 2002.
- [67] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-addressable Network," *Proceedings of ACM SIGCOMM*, 2001.
- [68] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-Aware Overlay Construction and Server Selection," *Proceedings of IEEE INFOCOM*, 2002.
- [69] M. Ripeanu, A. Iamnitchi, and I. Foster, "Mapping the Gnutella Network," *IEEE Internet Computing*, 2002.
- [70] Ritter, Why Gnutella Can't Scale. No, Really, <http://www.tch.org/gnutella.html>
- [71] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," *Proceedings of International Conference on Distributed Systems Platforms*, 2001.
- [72] S. Saroiu, P. Gummadi, and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *Proceedings of Multimedia Computing and Networking (MMCN)*, 2002.
- [73] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, "An Analysis of Internet Content Delivery Systems," *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, 2002.

- [74] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Network Support for IP Traceback," *ACM/IEEE Transactions on Networking*, 9(3), June, 2001.
- [75] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical Network Support for IP Traceback," *Proceedings of ACM SIGCOMM*, 2000.
- [76] M. T. Schlosser and S. D. Kamvar, "Availability and locality measurements of peer-to-peer file systems," *Proceedings of ITCOM: Scalability and Traffic Control in IP Networks*, 2002.
- [77] C. Schuba, I. Krsul, M. Kuhn, E. Spafford, A. Sundaram, and D. Zamboni, "Analysis of A Denial of Service Attack on TCP," *Proceedings of IEEE Symposium on Security and Privacy*, 1997.
- [78] S. Sen and J. Wang, "Analyzing Peer-to-peer Traffic Across Large Networks," *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, 2002.
- [79] C. Shields and B. N. Levine, "A Protocol for Anonymous Communication over the Internet," *Proceedings of 7th ACM Conference on Computer and Communication Security (ACM CCS)*, 2000.
- [80] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, B. Schwartz, S. T. Kent, and W. T. Strayer., "Single-Packet IP Traceback," *IEEE/ACM Transactions on Networking (ToN)*, Volume 10, Number 6, Pages 721-734, December, 2002.
- [81] D. X. Song and A. Perrig, "Advanced and Authenticated Marking Schemes for IP Traceback," *Proceedings of IEEE INFOCOM*, 2001.
- [82] K. Sripanidkulchai, The popularity of Gnutella queries and its implications on scalability, <http://www2.cs.cmu.edu/~kunwadcc/research/p2p/gnutella.html>
- [83] T. Stading, P. Maniatis, and M. Baker, "Peer-to-peer Caching Schemes to Address Flash Crowds," *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.
- [84] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," *Proceedings of ACM SIGCOMM*, 2001.
- [85] C. Tang and P. K. Mckinley, "On the Cost-Quality Tradeoff in Topology-Aware Overlay Path Probing," *Proceedings of the 11th IEEE Conference on Network Protocols (ICNP)*, 2003.
- [86] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger, "Network Topology Generators: Degree-Based vs. Structural," *Proceedings of In Proceedings of SIGCOMM'02*, 2002.

- [87] M. Waldvogel and R. Rinaldi, "Efficient Topology-aware Overlay Network," *Proceedings of ACM HotNets*, 2002.
- [88] C. Wang, L. Xiao, Y. Liu, and P. Zheng, "Distributed Caching and Adaptive Search in Multilayer P2P Networks," *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS)*, 2004.
- [89] Y. Xie and D. O'Hallaron, "Locality in Search Engine Queries and Its Implications for Caching," *Proceedings of IEEE INFOCOM*, 2002.
- [90] Z. Xu, C. Tang, and Z. Zhang, "Building Topology-aware Overlays Using Global Soft-state," *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS)*, 2003.
- [91] A. Yaar, A. Perrig, and D. Song, "Pi: A Path Identification Mechanism to Defend against DDoS Attacks," *Proceedings of IEEE Symposium on Security and Privacy*, 2003.
- [92] B. Yang and H. Garcia-Molina, "Comparing Hybrid Peer-to-Peer Systems," *Proceedings of the Twenty-first International Conference on Very Large Databases (VLDB)*, 2001.
- [93] B. Yang and H. Garcia-Molina, "Designing Super-peer Network," *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, 2002.
- [94] B. Yang and H. Garcia-Molina, "Efficient Search in Peer-to-peer Networks," *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, 2002.
- [95] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker, "On the Constancy of Internet Path Properties," *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, 2001.
- [96] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-resilient wide-area location and routing," Technical Report UCB//CSD-01-1141, U.C.Berkeley 2001.
- [97] Z. Zhuang, Y. Liu, L. Xiao, and L. M. Ni, "Hybrid Periodical Flooding in Unstructured Peer-to-Peer Networks," *Proceedings of International Conference on Parallel Processing*, 2003.

MICHIGAN STATE UNIVERSITY LIBRARIES



3 1293 02504 3617