1-1-2007

An analysis of structured and unstructured P2P systems over MANET

Cristian Negulescu
Ryerson University

AN ANALYSIS OF STRUCTURED AND UNSTRUCTURED P2P SYSTEMS OVER MANET

by

Cristian Negulescu

A project

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Engineering

in the Program of

Electrical and Computer Engineering

Toronto, Ontario, Canada, 2007

Declaration

I hereby declare that I am the sole author of this project.

I authorize Ryerson University to lend this project to other institutions or individuals for the purpose of scholarly research.

Signature:_____

December 2007

I further authorize Ryerson University to reproduce this project by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signature:_

December 2007

Abstract

An Analysis of Structured and Unstructured P2P Systems over MANET

Master of Engineering, 2007

Cristian Negulescu

Program of Electrical and Computer Engineering

Ryerson University

Peer-to-peer or P2P file-sharing application on wireless ad hoc mobile network (MANET) has gained a lot of interest in the recent years. A peer-to-peer network is an overlay network that is deployed over ad hoc networks. Our work is analyzing two P2P systems over MANET. The systems evaluated are representing two distinct categories in terms of network topology such as unstructured and structured architectures. RAON or Resource-Aware Overlay Network, which is an improvement of an existent P2P system called Gia, is an unstructured system. CAN or Content-Addressable Network is an existent P2P system in the category of structured architectures. Based on the simulations of the two P2P systems over MANET, we evaluated the performance in terms of query search success rate and query search delay.

Acknowledgements

I would like to thank my advisor Professor Muhammad Jaseemuddin for his support with the research and work that has been done throughout this project.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

File-sharing peer-to-peer (P2P) applications have grown in popularity and usage in the Internet in the last few years. They are based on P2P architecture that is characterized by direct access between peer computers rather than through a centralized server.

The popularity of P2P system comes from the possibility to share files between end users or peers over the Internet. The beginning of the P2P file-sharing systems started with Napster. The service provided by Napster [9] allowed users to exchange files based on information stored at a central server. The central servers are designed to store the central index to locate files available on the user systems. This centralized approach in searching the files proved to be not scalable. The evolution of P2P file sharing continued with a number of decentralized systems such as Gnutella [10] and Kazaa [17]. These systems form an overlay network in which each P2P node is capable of connecting to other nodes in the network.

Three categories of P2P systems have been identified based on the degree of interaction between peers by using one or more servers: decentralized (Gnutella, Freenet [18]), partially centralized (Kazaa, a more recent Gnutella) and hybrid of centralized and decentralized (Napster).

In case of highly dynamic decentralized P2P networks with complex topology, the degree of the placement of data in the overlay topology differentiates three categories of P2P systems: structured (Chord [12], CAN [11], Tapestry [14]), loosely structured (Freenet) and unstructured (Gnutella). Queries in these systems are not sent to a central site, but are instead distributed among the peers. Gnutella is the first system that uses an unstructured overlay network where the topology of the overlay network and placement of files within it is largely unconstrained.

The file search in the structured and unstructured overlay networks exhibits different performances. While the structured architecture is better suited for exact-match queries, the most common search operations on file-sharing systems are based on keyword search giving the unstructured system a greater advantage in such conditions. In case of unstructured overlay networks, the search is random over the network. In this case, the network is usually flooded with queries in lack of any information about the nodes that might store the relevant files. Upon receiving a query, each peer sends a list of the data content matching the query to the originating node. Because the load on each node grows with the total number of queries, which in turn

grows with system size, this approach is shown not feasible [5]. For structured network file searching is based on a defined and structured overlay topology with an exact mapping between files and the files' location. Thus, the queries can be more efficiently routed to the nodes storing the required files. For a very dynamic environment with a large number of nodes joining or leaving the network, the system must be able to maintain a high degree of structure stability required to route the information to the node with the target file. The structured systems will be less resilient for a very transient user environment because maintaining network information will be harder to achieve. A completely different effect is noticed in case of an unstructured network that can be easily adapted to fast changes in user network.

The P2P systems characteristics to consider for a scalable architecture should fulfill the following conditions: the possibility to have a keyword searching, a good replication over the nodes and adaptability to a highly transient node environment [29]. The P2P networks have been developed based on the normal wired Internet structure. But in the conditions of continuous changing communication environment with greater demand for mobility and easier access to a communication device, the wireless network brings new constraints to the P2P wireless network.

1.1 P2P and Mobile Networks

The fast development in wireless technology allows people to use devices such as cellular phones, personal digital assistants (PDAs) and laptops with great easiness and at the same time allows using of a large number of communication services consisting of e_mail transfers, music or pictures downloading, music playing, web browsing and text editing.

In wired network architectures the nodes are fixed and the network bandwidth will cover the traffic demands. In case of mobile hosts, the network is highly dynamic and unpredictable. The limitations of the wireless environment, processing power, battery life and bandwidth, are increasing the challenges in terms of network management. When there are no fixed structures in a network while all the nodes need to route the traffic themselves. That means that the traffic goes through multiple peers in the network before being received by the original target. This type of network is called multi-hop network. A mobile network based on multi-hop model with no central access point is called *Mobile Ad-hoc Network* or MANET.

P2P and MANET have received a lot of attention for research because they share many

similarities. Both types of network have similar particularities such as decentralized architectures, self-configurable and dynamic topology. P2P network is an overlay network, which is composed of nodes connected together forming a virtual network at the application level, independent of the underlying network. The goal of P2P file sharing system is to locate a set of servers that contain a given file. An ad-hoc network routing protocol role in MANET is to determine which route to take in order to reach a remote host. P2P file sharing application on a mobile ad-hoc network or MANET has appeared as a real choice in this context. The major challenge in the integration of the two previous mentioned architectures would be to locate the desired content efficiently and deliver the content to the destination reliably under the MANET environment.

The file sharing application is only one of the P2P applications over MANET. Another MANET application is a wireless sensor network. There are also other domains for implementing this architecture, such as military defense network, emergency personnel and the police. The real driver behind P2P at this moment proves to be downloading and sharing music files. Thus, in the case of a P2P over MANET network, each peer can access the files of other peers in the network and share the personal files.

1.2 Project Outline

The goal of the project is to present a comparison of two architectural choices of P2P systems namely structured and unstructured, and to show how these architectures can be adapted in the MANET environment. The project will focus on the challenges of adaptation of structured and unstructured P2P architectures to the variable environment of a mobile environment such MANET.

There will be analyzed existing P2P architectures with their particularities related to the interaction between peers and network topology. A presentation of the categories of P2P systems will describe the differences among the existent P2P systems. Another point of interest will be the integration of the P2P systems with MANET and challenges occurred in this integration process. The main application proposed as subject of analysis for P2P over MANET is P2P file sharing system.

The report is organized as follows. Chapter 2 will provide background information on

P2P file sharing systems and their applicability over MANET environment. In Chapter 3, we discuss and analyze the designs and differences between two proposed P2P file sharing systems over MANET, RAON and CAN. The evaluation of the two systems mentioned before is presented in Chapter 4 through simulations results. Chapter 5 summarizes the comparison results and the performance of both RAON and CAN systems over MANET.

Chapter 2 Background

In the following chapter there will be presented the P2P file sharing systems and MANET routing architecture. The P2P systems will be categorized based on degree of centralization and network topology. The routing MANET protocols will be analyzed and described in order to give an overview on the MANET architecture.

2.1 P2P File Sharing Architecture

Peer-to-peer (P2P) is an overlay network with distributed systems without any hierarchical organization or centralized control. Peers form self-organizing overlay networks that are overlaid on the Internet Protocol (IP) networks. A P2P network offers various features such as robust wide-area routing architecture, efficient search of data items, selection of neighbor peers, redundant storage, hierarchical naming, scalability, and fault tolerance. Peer-to-peer overlay systems are providing roles where a client may also be a server. It allows access to its resources by other systems and supports resource sharing. One of the most popular applications of P2P is file sharing. Over the next section the focus will be on describing P2P file sharing system.

Typically, P2P file sharing applications allow users to share their files with other end-users, which are referred as peers. Peers join together to form a network, and they can find the desired files and download them directly from other peers. Each peer is a server as well as a client. Nodes that have downloaded the content become a content server themselves. This is different from the client-server model where one server serves all clients. Different P2P systems are distinguished by the degree of centralization and the network structure.

2.1.1 P2P Classification

The P2P systems can be split in three categories based on the mode in which interaction between peers is controlled.

The first category is represented by **purely decentralized** P2P architectures with examples such as Gnutella in its original variant and Freenet. All nodes in the network perform

exactly the same tasks, acting both as servers and clients without central coordination of their activities. The nodes are also called "servents" or SERVers + clieENTS.

The second category is represented by **partially centralized** systems such as KaZaa and a more recent version of Gnutella. In this case, some of the nodes assume a more advanced role than the rest of the nodes, acting as local central indexes for files shared by local peers. These nodes are called SuperNodes or SuperPeers. The nodes with sufficient resources in terms of CPU, memory and bandwidth are elected as SuperPeers. These SuperNodes do not constitute single points of failure for a P2P network because they are dynamically assigned and in case they are subject to failure or malicious attack the network will take action to replace them with others.

The third category is **hybrid decentralized** architecture. The main example is also the first P2P file sharing system such Napster. A central server facilitates the interaction between peers by maintaining indexes of the shared files stored on the nodes existent in the network. The end-to-end interaction is between two peer clients. However, these central servers allow for this interaction by performing the lookups and identifying the nodes of the network where the files are located. The central server will represent the single point of failure for these architectures. This makes them vulnerable to technical failure, malicious attack, and law enforcement.

The P2P topology creates an overlay network, which is totally unrelated to the physical network that connects the different nodes. A network structure in P2P context is described by the content location with respect to the network topology. We have three categories related to the network structure.

In the first category, **unstructured** networks such as Gnutella, the placement of files is completely unrelated to the overlay topology. Since there is no information about which nodes have the relevant files, searching is based on flooding the network with queries in which various nodes are probed to find if they have any files matching the query. The advantage of such systems is that they can easily accommodate a highly transient node population. The disadvantage is that it is hard to find the desired files without distributing queries widely. For this reason unstructured P2P systems are considered to be unscalable.

In the second category, **structured** networks such as Chord, CAN, Kademlia [6], Tapestry, have emerged in an attempt to address the scalability issues that unstructured systems encountered. The random search methods adopted by unstructured systems seem to be unscalable [20]. The structured systems were proposed to address scalability issues. In these

structured systems, the overlay network topology is tightly controlled and files are placed at precisely specified locations. These systems provide a mapping between the file identifier and location, in the form of a distributed routing table named distributed hash table (DHT). In this way the queries can be efficiently routed to the node with the desired file [20]. The structured systems offer a scalable solution for exact-match queries, queries in which the complete identifier of the requested data object is known. The disadvantage of structured systems is that it is hard to maintain the structure required for routing in a very transient node population, in which nodes are joining and leaving at a high rate [29].

The third category, **loosely structured** networks such as Freenet are situated in between the two categories mentioned above. File locations are affected by routing hints, but they are not completely specified, so not all searches succeed [8].

2.1.2 Distributed Hash Tables

The main goal of our work will be the comparison between structured systems and unstructured systems. In structured systems the Distributed Hash Tables or DHT are the main components.

Distributed Hash Tables are algorithms used in peer-to-peer applications, required to provide a reliable, scalable, fault tolerant and efficient way to manage P2P networks. Hash tables are data structures that can be used to map keys to values, store (key, value) pairs, and retrieve values using the given keys. Distributed hash tables are similar to distributed data structures that are used in P2P applications to store and retrieve data efficiently. In a classic hash table, the objects are stored in different buckets according to each object's hash value, which is obtained by applying a hash function to the data being stored. DHTs store the values on a number of hosts instead of just a single host. In DHT P2P networking, every node in the network has a globally unique identifier (GUID). Each node functions as a container for the distributed data. Nodes in this type of P2P network form an overlay network, which controls how the nodes are connected and how many neighbors a node can connect to. An overlay network is a virtual network that runs on top of another network and can be managed independently. An overlay network can utilize the underlying protocols. Some DHTs prefer using connectionless UDP whereas others utilize the connection oriented TCP. In P2P world, the overlay network functions at the

application level. DHTs provide decentralized operation, eliminating the need for a centralized server to control the P2P network. Load is uniform balanced among the peer nodes in the network. The system is based on the assumption that the network is not static and changes occur frequently with nodes joining and leaving the network.

Short descriptions of the most important DHT algorithms are presented in next section. We start with **Chord** [12], a DHT algorithm based on (key, values) pairs. Chord resembles a ring from the network topology's point of view. In the overlay network, the ID of the P2P network client represent the key and the corresponding value is the data that is stored at the corresponding nodes. In Chord, each node stores only a subset of all the keys, which increases the reliability of the network. Chord can be used in implementing different services, including distributed information lookup services. Scalability is one of the characteristics that Chord tries to achieve. The system can still function efficiently when the size of the network grows. Availability also is important in order for a system to function even when the network is partitioned or some nodes fail. The (key, value) pairs have to be evenly distributed ensuring that the system is load-balanced. When the system is under high demand from nodes joining and leaving the network, it is possible for the overlay network to become partitioned. This happens when sufficiently many nodes leave the network simultaneously almost splitting the network in two separate subnets that have severely limited access to the other. Chord has been designed so that these kinds of network partitions do not render the whole network unusable. Chord also maintains a location table in each node. This table functions as a cache and contains the recent nodes that Chord has discovered while running. The cache is a routing table that maps node identifiers to their corresponding IP-addresses and ports. This cache can be used to speed up key lookups. At the application level, Chord is divided into Chord server and Chord client, which both run on the same network node.

Tapestry [14] considers the network distance when looking up keys. Tapestry uses neighbor maps and tree-like routing to deliver the messages to target nodes. When a node sends a lookup, it transmits it to the addresses that are found in its local neighbor map, which contains (ID-number, IP-address) pairs. The node always forwards messages to those target nodes that are topologically closer to it. Tapestry is a P2P overlay routing algorithm that can be used to implement distributed applications. Tapestry provides Decentralized Object Location and Routing in an efficient, scalable and location independent manner.

Kademlia [6] decides where to store particular (key, value) pairs by doing a bitwise XOR operation on the 160-bit node id numbers and finds out the topological distance between them. Lookups are then forwarded to the nearest available node. Kademlia protocol states that each node must know at least one node in each of its subtrees. This way every node can be reached from any given node using node's ID. Kademlia is a widely used algorithm that utilizes UDP protocol in communication. Kademlia algorithm has been implemented in eMule, mlDonkey and Overnet file sharing applications for indexed file searching.

CAN [11] or Content Addressable Network is the last DHT algorithm presented in this section. CAN's topology is a d-dimensional Cartesian coordinate space. The coordinate space is a logical one and is not associated with a physical coordinate space. It is always entirely partitioned among all the nodes in the system, each node owning part of it. When a new node joins the network overlay, an available partition is split in half to allocate space for the new node. Node is considered another node's neighbor if their coordinate spaces meet along d - 1 axis. In 2-dimensional case, the partitions in coordinate space would have to meet along one axis. Each node in the CAN system stores a part of the hash table and in addition also keeps information about the neighbor nodes that are the closest to it. The number of hops gives the closest neighbor. CAN could be used to implement large scale storage management and retrieval systems. One test implementation of CAN is known as pSearch, which is a semantic and content based search service that utilizes a hierarchical variant of CAN, eCAN [25]. We will discuss in more detail about CAN in the next chapter where the properties of this algorithm will be analyzed.

The main performances issues occurred for the DHT-based systems are represented by four elements: churn, packet loss, proximity, caching. In P2P applications, *churn* is a term that refers to the rate by which the overlay network changes or the rate of node arrivals and departures from the network. The way in which the applications handle these changes has a big impact on the system overall performance. It has been argued that many existing DHT systems cannot handle high churn rates well [38]. Churn can be measured by the node "session time", which is the time from node join to node departure. The lower the average "session time" for network nodes, the higher the churn. Node lifetime, on the other hand, is the time from first join to final departure. Packet loss refers to the percentage of dropped packets from total amount sent. Packet loss depends on the underlying network conditions. However, the DHT system should

handle occasional high packet loss rate without rendering the service unusable. In DHT based systems, packet loss is often a symptom of network congestion. Proximity routing [1] is used in several DHTs to improve lookup times. When doing proximity routing, queries are directed to nodes that are geographically close to originating node to reduce the latency that is caused by message transfer in the network. CAN, Kademlia, Tapestry and Chord all use proximity routing in their latest versions. Caching has been successfully used on various different applications and the same principle can be applied also to DHTs to improve their performance. There are two different caching schemes: passive caching and proactive caching [39]. In passive caching, a copy of the requested node is stored in every node on the route from object storage node to querying node. In proactive caching, a small subset of the most popular resources is cached in every node beforehand. This can lower the average retrieval delay to 1 hop. DNS, web accessibility and multimedia content distribution on the Internet could all benefit from proactive replication and caching of resources [39].

The metrics used in the evaluation of the DHT systems are the number of hops, latency and stabilization interval. Number of hops or hop-count is an indicator that is used when measuring distances between nodes. A hop represents forwarding of a message from one network node to another. Latency is an indicator that is used when measuring key lookup delay. Latency is the same as the round trip time for a query to come back to the originating node. Stabilization means removing routing data for nodes that have either left the network or dropped out. Stabilization interval indicates how often this is done.

2.2 MANET Architecture

Many protocols have been developed for mobile ad hoc networks, with the goal of achieving efficient routing. Such protocols must deal with typical limitations of the MANET networks due to dynamic topology, high power consumption and high error rates. The routing protocols used in these environments should have some features as: minimum hop and delay to destination, fast recovery to link changes, stable route selection, distributed operation and loop avoidance. The next discussion will focus on some of the existing routing protocols proposed for MANET. The description of the AODV (Ad-hoc On-demand Distance Vector) routing protocol will follow as the chosen routing protocol used in our P2P over MANET architectures

comparison.

The characteristics required for MANET environment are described in RFC 2501 [40]:

1. Dynamic topologies: Nodes can move arbitrarily with respect to other nodes in the network.
2. Bandwidth-constrained: Nodes in an ad hoc network are mobile. Thus, they are using radio links that have far lower capacity than hardwired links could use.
3. Energy constrained operation: Mobile nodes are relying on batteries. A primary design criteria based on energy conservation is important in this case.
4. Limited physical security: In general, radio networks are vulnerable to physical security threats compared to fixed networks. The possibility of eavesdropping, spoofing and denial-of-service attacks is higher. Existing link security techniques can be applied.

The properties mentioned above for MANET architecture impose some essential features necessary in a proper functioning of the ad hoc routing protocols:

Distributed operation: Ad hoc networks are targeted to operate in distributed manner.

Loop-freedom: It refers to avoiding packets spinning around in the network for arbitrary time.

Demand based operation: Ad hoc routing does not have to assume uniform traffic load in a network but it can adapt to traffic patterns on need basis. This will increase route discovery delay but when implemented properly bandwidth and energy resources can be more efficiently utilized.

Proactive operation: This is opposite to demand based operation. If additional delays that occur in demand based operation are unacceptable, proactive approach can be used especially when energy and bandwidth capacities support the use of proactive operation.

Security: Because of the vulnerabilities in the physical security mentioned before, ad hoc routing protocols are exposed to multiple types of attacks. Routing protocols security is desirable.

Unidirectional link support: Many routing algorithms require bidirectional links to be capable of functioning. However, unidirectional links are more general in radio networks. It is desirable that ad hoc routing protocols can handle a situation where two oppositely directed unidirectional links form the only bidirectional connection between the nodes.

The MANET routing protocols differ in the approach used for searching a new route or modifying an existent one based on hosts mobility. There are three types of protocols: proactive, reactive and hybrid.

The proactive or table-driven routing protocols maintain consistent, up-to-date routing information from each node to every other node in the network. These protocols require each

node to maintain one or more tables to store routing information. They respond to changes in network topology by propagating updates throughout the network in order to maintain a consistent network view. The areas in which they differ are the number of necessary routing-related tables and the methods by which changes in network structure are broadcast. The disadvantage of the proactive approach is that the amount of routing related traffic may waste a large portion of the wireless bandwidth, especially for protocols that use periodic updates of routing tables.

In case of reactive routing or source-initiated on-demand approach, the protocols create routes only when desired by the source node. When a node requires a route to a destination, it initiates a route discovery process within the network. This process is completed once a route is found or all possible routes have been examined. Once a route has been established, the route is maintained by a route maintenance procedure until either the destination becomes inaccessible or the route is no longer needed.

The hybrid approach combines proactive and reactive approaches in the routing protocol. In this type of routing protocols, the network is divided into zones. The scope of the proactive procedure is limited to the same zone and reactive routing is used when the source and destination are not in the same zone.

2.2.1 DSDV

In the category of proactive routing protocols, as an example we present Destination Sequence Distance Vector or DSDV [4]. DSDV is a distance vector routing protocol. It is based on the distributed Bellman-Ford routing algorithm. It works on hop-by-hop basis meaning that every node maintains a routing table that contains next-hop entry and the number of hops needed for all reachable destinations. DSDV uses a concept of *sequence numbers*. The sequence number is originated from the destination node. DSDV requires nodes to *broadcast* periodical route advertisements. The advertisement contains the routing table entries of the advertising node. These entries contain the address of destination, next hop and hop count to that destination and the last known sequence number originated by that destination. When a node receives an advertisement it updates its routing table on this basis. Routes with greater sequence numbers are always preferred. If the sequence numbers are equal, a route with lower hop count is chosen. The

receiving node increases the hop counts in the advertisement since the destination needs one hop more to be reached. The receiving node will pass this new information forward within its own route advertisement. When a node detects link failure it marks all routes through that link with hop count of infinity and broadcasts update information. Because frequent route advertisements can generate a lot of control traffic, DSDV introduces two types of route update packets. The first is known as *full dump* containing all available routing information and may require several *network protocol data units (NPDUs)*. The second has smaller *incremental* packets that are used to distribute only information that has changed since last full dump.

2.2.2 DSR and AODV

In case of reactive routing protocols, we exemplify Dynamic Source Routing or DSR and Ad-Hoc On-Demand Distance Vector or AODV. DSR uses *source routing* to send packets [7]. DSR only sends route request when it needs one and does not require that the nodes maintain routes to destinations that are not communicating. It uses source routing, which means that the source must know the complete hop sequence to the destination. Each node maintains a route cache, where all routes it knows are stored. The route discovery process is initiated only if the desired route cannot be found in the route cache. To limit the number of route requests propagated, a node processes the route request message only if it has not already received the message and its address is not present in the route record of the message. The source determines the complete sequence of hops that each packet should traverse. This requires that the sequence of hops is included in each packet's header. A negative effect of this is the routing overhead every packet has to carry. However, one big advantage is that intermediate nodes can learn routes from the source routes in the packets they receive. Because finding a route is generally a costly operation in terms of time, bandwidth and energy, this is a strong argument for using source routing. Also, source routing avoids the need for up-to-date routing information in the intermediate nodes through which the packets are forwarded because all necessary routing information is included in the packets. The routing algorithm avoids routing loops because the complete route is decided by a single node and is not based on a hop-by-hop decision.

DSR uses two basic mechanisms called *Route Discovery* and *Route Maintenance*. *Route Discovery* is used whenever a source node requires a route to a destination node. The source

node looks up its route cache to determine if it already contains a route to the destination. If the source finds a valid route to the destination, it uses this route to send its data packets. If the node does not have a valid route to the destination, it initiates the route discovery process by broadcasting a Route Request message. The Route Request message contains the address of the source and the destination, and a unique identification number. An intermediate node that receives a route request message searches its route cache for a route to the destination. If no route is found, it appends its address to the route record of the message and forwards the message to its neighbors. The message propagates through the network until it reaches either the destination or an intermediate node with a route to the destination. If the receiving node has seen a request from the same initiator with the same id or if its address is already in the *route record* of the Route Request packet, it discards the packet. Otherwise the receiving node adds it own address to the route record of the request and broadcasts the request forward. After the Route Discovery process is completed, route record of a Route Request contains a complete source route from the initiator to the target. This information is sent back in a Route Reply message. The reply is sent back to the source based on a route in replier's cache or it is piggybacked on a Route Request packet for the initiator. *Route Maintenance* is used to handle route failures. When a node encounters a fatal transmission problem at its data link layer, it removes the route from its route cache and generates a Route Error message. The Route Error message is sent to each node that has sent a packet routed over the broken link. When a node receives a Route Error message, it removes the hop in error from its route cache. Acknowledgment messages are used to verify the correct operation of the route links. In wireless networks, acknowledgments are often provided as a standard part of the MAC protocol in use such as the link-layer acknowledgment frame defined by IEEE 802.11. If a built-in acknowledgment mechanism is not available, the node transmitting the message can explicitly request a DSR-specific software acknowledgment.

AODV [8] combines some properties of both DSR and DSDV. It uses *route discovery process* to handle the routes changes. The protocol is seen as a *reactive* one. In the same time, it adopts DSDV hop-by-hop routing tables for maintaining routing information. Each route entry in the routing table stores the hop count, next hop, and the greatest sequence number of the destination. There is also an expiry time to indicate the validity of each route. The route entry is removed from the routing table if it is not used or updated within that time. It does not need to maintain routes to nodes that are not communicating. The reactive property of the routing

protocol implies that it only requests a route when it needs one and does not require that the mobile nodes maintain routes to destinations that are not communicating. AODV guarantees loop-free routes by using sequence numbers that indicate how new a route is. AODV requires each node to maintain a routing table containing one route entry for each destination that the node is communicating with. For *Route Discovery* whenever a source node desires a route to a destination node for which it does not already have a route, it broadcasts a *route request* (RREQ) message to all its neighbors. The neighbors update their information for the source and create *reverse route entries* for the source node in their routing tables. A neighbor receiving a RREQ may send a *route reply* (RREP) if it is either the destination or if it has an unexpired route to the destination. If any of these two cases is satisfied, the neighbor unicasts a RREP back to the source. Along the path back to the source, intermediate nodes that receive the RREP create *forward route entries* for the destination node in their routing tables. If none of the two cases mentioned is satisfied, the neighbor forwards the RREQ.

Each mobile node keeps a cache where it stores the source IP address and ID of the received RREQs. If a mobile node receives another RREQ with the same source IP address and RREQ ID during this period, it is discarded. Hence, duplicated RREQs are prevented and not forwarded. *Route Maintenance* is necessary when a link in a route fails. Therefore the node upstream of the failure point invalidates all its routes that use the broken link. Then, the node broadcasts a *route error* (RERR) message to its neighbors. The RERR message contains the IP address of each destination, which has become unreachable due to the link failure. Upon reception of a RERR message, a node searches its routing table to see if it has any routes to the unreachable destinations, which use the originator of the RERR as the next hop. If such routes exist, they are invalidated and the node broadcasts a new RERR message to its neighbors. This process continues until the source receives a RERR message. The source invalidates the listed routes as previously described and reinitiates the route discovery process if needed.

A Local Repair feature is supported in a newer version of AODV. If the intermediate node that detected the link failure is closer to the destination than the source, it would initiate a RREQ and try to resolve a new route locally. Any data packet received in the meantime is queued at the node. If a new route to the destination is discovered, then the node flushes all the queued packets. If a new route cannot be found, it sends an RREP message along the reverse path and all the data queued at this node are dropped.

The difference between AODV and DSR is how the next hop routing information is determined. In AODV, this information is stored at the associated nodes on the forwarding route, and thus, routing decisions are made independently at each node. In DSR the complete hop sequence is stored in the header of all data packets. Routing decisions are solely made by the source. Thus, if an intermediate node found a better and more up-to-date route to the destination than the one known to the source, AODV would be able to take advantage of that and DSR would not. Because reactive routing protocols can conserve more bandwidth and energy than proactive ones, AODV will be the routing protocol considered for our comparison analysis.

2.2.3 ZRP

Zone Routing Protocol or ZRP [41] is a hybrid protocol that is part proactive and part reactive. The proactive part, uses a modified distance vector scheme within the *routing zone* of each node. The routing zone is determined by a *zone radius* that is the minimum number of hops it should take to get to any node. Thus, each node has a routing zone, which is composed of nodes within its local area. This proactive component is called *Intrazone Routing Protocol* (IARP). The reactive component is called *Interzone Routing Protocol* (IERP), and uses queries to get routes when a node is to send a packet to a node outside of its routing zone.

ZRP uses a method called *bordercasting* in which a node asks all nodes on the border of its routing zone to look for the node outside of its routing zone. The Intrazone Routing Protocol (IARP) proactively maintains routes to destinations within a local neighborhood, which is referred to as a routing zone. The node's routing zone is a collection of nodes whose minimum distance in hops from the specified node is no greater than the zone radius. Note that each node maintains its own routing zone. The consequence is that the routing zones of neighboring nodes overlap. The operation of the reactive Interzone Routing Protocol (IERP) is similar to route discovery process of reactive routing protocols. An IERP route discovery is initiated when no route is locally available to the destination of a data packet. The source generates a route query message, which is uniquely identified by a combination of the source node's address and request number. The query is then relayed to a subset of neighbors as determined by the bordercast algorithm. Upon receipt of a route query message, a node checks if the destination lies in its zone or if a valid route to it is available in its route cache. If the destination is found, a route reply is

sent back to the source. If not, the node "bordercasts" the query again. Since the topology of the local zone of each mobile node is known, global route discovery is simplified. ZRP uses a concept called *bordercasting* in order to broadcast a route query from neighbor to neighbor. Bordercasting means that the route query is directed toward regions of the network that have not yet been covered by the query. A covered node is the one that belongs to the routing zone of a node that has received a route query. The route query traffic is reduced by directing route queries outwards from the source and away from covered routing zones [41].

Chapter 3 Analysis of Unstructured and Structured P2P Systems over MANET

In the next chapter an analysis of unstructured and structured P2P systems will be completed based on two P2P systems proposed for MANET: RAON (Resource–Aware Overlay Network) and CAN (Content-Addresable Network). Both structured and unstructured P2P systems have advantages and disadvantages. In the first part of the chapter, the unstructured P2P systems such as Gnutella, Gia and improvements brought by RAON to Gia, are presented. In the second part, as the structured P2P system proposed for ad-hoc mobile environment, CAN is presented.

3.1 Gia P2P System

The centralized systems such as Napster have been replaced by new *decentralized* systems such as Gnutella [10] that distribute both the download and search capabilities. These systems establish an overlay network of peers. Queries are not sent to a central site, but are instead distributed among the peers. Gnutella, the first of such systems, uses an *unstructured* overlay network in that the topology of the overlay network and placement of files within it is largely unconstrained. The network flooding with the queries and answers for the content matches have raised questions about Gnutella scalability. Following Gnutella's approach, several other decentralized file-sharing systems such as KaZaA [17] have become popular. KaZaA is based on the proprietary Fasttrack technology, which uses specially designated *SuperNodes* that have higher bandwidth connectivity. Pointers to each peer's data are stored on an associated SuperNode, and all queries are routed to SuperNodes. Although some Gnutella clients now implement the SuperNode proposal, its scalability remains a challenge.

A new P2P file-sharing system, called Gia [20], is trying to implement some features to improve the performance of Gnutella. Like Gnutella and KaZaA, Gia is decentralized and unstructured. As in the recent work by Lv *et al.* [30], Gia replaces Gnutella's flooding with random walks. Based on Adamic *et al.* [22], Gia addresses the issue of the overlay network's topology while using random walks and therefore includes a topology adaptation algorithm. The lack of flow control has been addressed in the original Gnutella design [10], and Gia also introduces a token-based flow control algorithm. Like KaZaA, Gia takes into account the

heterogeneity in peer bandwidth. Resource-Aware Overlay Network or RAON [37] uses Gia as the foundation for its design. RAON adds the features required to address issues of adaptability of Gia in MANET. RAON will be the unstructured algorithm analyzed in our simulations.

3.1.1 Gnutella P2P System

In its original architecture, Gnutella is built at the application level as a virtual overlay network with its own routing mechanisms. The network can be seen as a distributed file storage system, allowing its users to specify files on their machines, which they want to share with other peers. Although the Gnutella protocol supports a traditional client/centralized server search algorithm, Gnutella is a decentralized model for document location and retrieval, as shown in Figure 3.1.

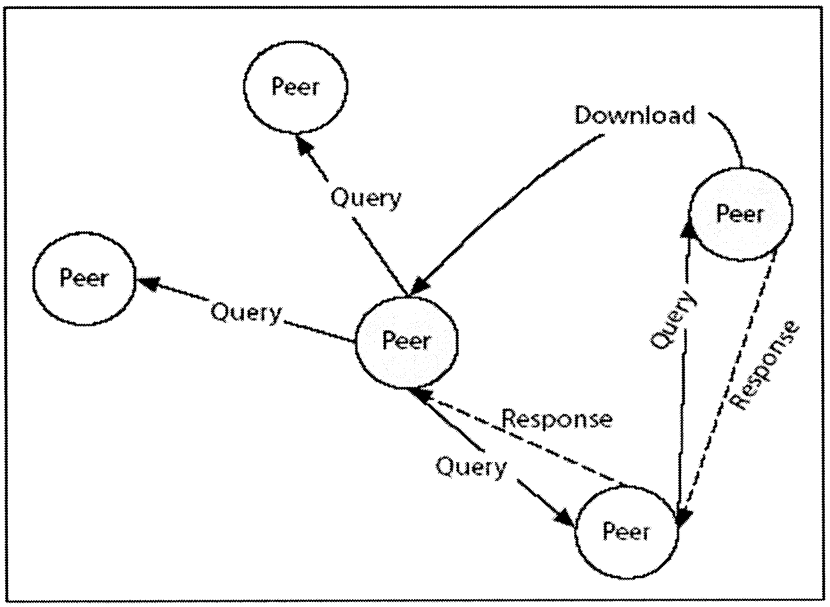

Figure 3.1 Gnutella decentralized architecture

In this model, every peer is a server or client, and is also referred to as a *servent*. This system is neither a centralized directory nor does it possess any precise control over the network topology or file placement. Once the nodes are joining the network the resulting topology has certain properties, but the placement of data items is not based on any knowledge of the topology as in the structured P2P designs. To locate a data item, a peer queries its neighbors, and the most typical query method is flooding. Such design is extremely resilient to peers entering and leaving

the system. However, the current search mechanisms are not scalable and generate unexpected loads on the network. These peers are responsible for managing the background traffic that spreads the information used to maintain network integrity. Gnutella uses IP as its underlying network service, while the communication between hosts is specified in a form of application level protocol supporting four types of messages:

PING and PONG Messages: A peer joining the network initiates a broadcasted PING message to announce its presence. The PING message is then forwarded to its neighbors and initiates a PONG message as reply, which contains information about the peer, such as the IP address, number and size of files shared.

QUERY and QUERY HITS Messages: QUERY contains a user specified search string and the minimum speed requirements of the responding host. QUERY HITS messages are replies to QUERY messages and include the IP and port and speed of the responding host, the number of matching files found and their indexed result set.

GET and PUSH Messages: File downloads are performed directly between two peers using these types of messages.

Due to its distributed nature, a network of servents that implement the Gnutella protocol is highly fault-tolerant because operation of the network will not be interrupted if a subset of servents goes offline. To join the system, a new servent initially connects to one of several known hosts that are available. Once connected to the network, peers send PING messages to interact with each other. The nodes send back a PONG message to identify themselves and propagate the PING message to their neighbors. Each message has a randomly generated identifier. Each peer keeps a short memory of the recently routed messages used to prevent re-broadcasting and to implement back-propagation. Messages are flagged with TTL and number of hops fields. To become a member of the network, a servent or peer has to open one or many connections with other peers that are already in the network. To cope with the unreliability after joining the network, a peer periodically sends PINGs to its neighbors to discover other participating peers. Peers decide where to connect in the network based only on local information. Thus, the entire application-level network has servents as its peers and open TCP connections as its links, forming a dynamic, self-organizing network of independent entities.

In unstructured systems such as Gnutella, the nodes broadcast random query messages to identify in which hosts the files may be located. A host in Gnutella uses TTL-limited flooding or

broadcast to distribute Ping and Query messages to its neighbors. The response messages are routed back along the opposite path through which the original request arrived. To limit the spread of messages through the network, each message header contains a time-to-live (TTL) field. At each hop the value of this field is decremented, and when it reaches zero the message is dropped. Since the response messages contain the same ID as the original messages, the host checks its routing table to determine along which link the response message should be forwarded. In order to avoid loops, the nodes use the unique message identifiers to detect and drop duplicate messages. This technique improves efficiency and preserves network bandwidth. Once a node receives a Query Hit message, indicating that the target file has been identified at a certain node, it initiates a direct download, establishing a direct connection between the source and target node. The disadvantage of Gnutella is that the TTL effectively segments the Gnutella network into subnets. If the TTL were removed, the network would be flooded with requests. Gnutella generates also a considerable amount of query traffic on the networks. This suggests that the Gnutella network is faced with a scalability problem [12].

In the most recent versions of Gnutella it is introduced the notion of SuperNodes or SuperPeers to address the scalability issue. This concept was first proposed in the FastTrack architecture [17], which is better known by the popular client names KaZaA and Morpheus. The nodes are dynamically assigned the task of servicing a small subpart of the peer network by indexing and caching files contained in the part of the network they are assigned to. Nodes with sufficient resources in terms of bandwidth, memory, and CPU power are automatically selected as SuperPeers and establish connections with other SuperPeers, forming a flat unstructured network of SuperPeers. When a node wants to locate a file, it simply sends a query to its SuperPeer, and if the SuperPeer finds the file index locally, it sends a reply back to the node. Otherwise, it forwards or broadcasts the query to other SuperPeers. Thus, the SuperPeer acts like a local server. The SuperPeer also sets the number of clients required for it to remain a SuperPeer. If it receives at least the required number of connections to client nodes within a specified time, it remains a SuperPeer. Otherwise it turns into a regular client node. If no other SuperPeer is available, it tries to become a SuperPeer again for another probation period. This architecture is sometimes referred as partially decentralized unstructured P2P system, while the original Gnutella is classified as purely decentralized unstructured P2P system.

Saroiu *et al.* [24] examined the bandwidth, latency, availability, and file sharing patterns

of the peers in Gnutella and Napster, and outlined the existence of significant heterogeneity in both systems. Chawathe *et al.* [20] proposed a model called Gia, by modifying Gnutella's algorithm to include flow control, dynamic topology adaptation, one-hop replication, and consider peer heterogeneity.

3.1.2 Gia Design

Gia system appeared as a necessity to improve Gnutella scalability issues. The Gia's goal is to create a P2P system that can handle high query rates and function properly with increasing system sizes, but in the same time to avoid overloading the nodes due to limited capacity. The problem of Gnutella sits in its scalability due to the flooding mechanism used in the search protocol. From previous section of the chapter we know that Gnutella uses TTL limited flooding to distribute its queries. Based on results from Lv *et al.*[30] and Cohen *et al.* [42], two methods such as multiple parallel random walks and proactive replication algorithm are proposed to avoid flooding. To search for an object by using a random walk, a node chooses a neighbor at random and sends the query to it. Each neighbor in turn repeats this process until the object is found. Random walks avoid the problem of the exponentially increasing number of query messages that arise in flooding. In case of the optimal replication scheme for random search, the objects are replicated proportional to the square-root of their query rate. This replication scheme can be implemented in a distributed mode by replicating objects a number of times proportional to the length of the search. These combined approaches, proactive replication plus parallel random walking [30], proved a significant improvement in performance for flooding and passive replication as a measure of number of overlay hops and per-node query load. But, the problem of scalability is not fully addressed if the node overloading due to their capacity constraints is not considered.

Gia design tries to incorporate the methods to reduce flooding, but in the same time takes into consideration the capacity constrains associated with each node in the P2P network. The capacity of a node depends upon a number of factors including its processing power, disk latencies and access bandwidth. In the design of Gia, the network heterogeneity and client capacity are taken into account in order to achieve a fully scalable architecture. It is assumed that client capacity is a quantity that represents the number of queries that the client can handle per

second. In practice, the capacity will have to be determined as a function of a client's access bandwidth, processing power and disk speed. Gia's design is based on four components:

- *topology adaptation* protocol: a protocol that puts most nodes within short reach of high capacity nodes. The adaptation protocol ensures that the well-connected or high-degree nodes, which receive a large proportion of the queries, actually have the capacity to handle those queries.
- *flow control* scheme: a scheme to avoid overloaded hot-spots. The flow control protocol explicitly acknowledges the existence of heterogeneity and adapts to it by assigning flow-control tokens to nodes based on available capacity.
- *one-hop replication:* replication of pointers to content. All nodes maintain pointers to the content offered by their immediate neighbors. Since the topology adaptation algorithm ensure the association between high capacity nodes and high degree nodes, the one-hop replication guarantees that high capacity nodes are capable of providing answers to a greater number of queries.
- *search protocol:* a protocol based on biased random walks that directs queries towards high-capacity nodes, which are typically best able to answer the queries.

The *topology adaptation* algorithm is the core component that connects the Gia client to the rest of the network. When a node starts up, it uses bootstrapping mechanisms similar to those in Gnutella to locate other Gia nodes. Each Gia client maintains a *host cache* consisting of a list of other Gia nodes (their IP address, port number, and capacity). The goal of the topology adaptation algorithm is to ensure that high capacity nodes are indeed the ones with high degree and that low capacity nodes are within short reach of higher capacity ones. To achieve this goal, each node independently computes a *level of satisfaction* (S). This is a quantity between 0 and 1 that represents how satisfied a node is with its current set of neighbors. A value of S = 0 means that the node is quite dissatisfied, while S = 1 suggests that the node is fully satisfied. As long as a node is not fully satisfied, the topology adaptation continues to search for appropriate neighbors to improve the satisfaction level. Thus, when a node starts up and has fewer than some pre-configured minimum number of neighbors, it is in a dissatisfied state (S = 0). As it gathers more neighbors, its satisfaction level rises, until it decides that its current set of neighbors is sufficient to satisfy its capacity. To add a new neighbor, a node, called X, randomly selects a small number of candidate entries from those in its host cache that are not marked dead and are

not already neighbors. From these randomly chosen entries, X selects the node with maximum capacity greater than its own capacity. If no such candidate entry exists, it selects one at random. Node X then initiates a three-way handshake to the selected neighbor, called Y. During the handshake, each node makes a decision whether or not to accept the other node as a new neighbor based upon the capacities and degrees of its existing neighbors and the new node. In order to accept the new node, we may need to drop an existing neighbor. If, upon accepting the new connection, the total number of neighbors would still be within a max_nbrs value, then the connection is automatically accepted. Otherwise, the node must see if it can find an appropriate existing neighbor to drop and replace with the new connection.

The *flow control scheme* used by Gia avoids creating hot-spots or overloading a node. The scheme allows a sender to direct queries to a neighbor only if that neighbor has notified the sender that it is willing to accept queries from the sender. Gia uses random walks to address scaling problems with flooding by forwarding a single copy of each query. To provide better flow control, each Gia client periodically assigns flow-control tokens to its neighbors. Each token represents a single query that the node is willing to accept. Thus, a node can send a query to a neighbor only if it has received a token from that neighbor, thus avoiding overloaded neighbors. A node allocates tokens at the rate at which it can process queries. If it receives queries faster than it can forward them either because it is overloaded or because it has not received enough tokens from its neighbors, then it starts to queue up the excess queries. If this queue gets too long, it tries to reduce the incoming flow of queries by lowering its token allocation rate. To provide an incentive for high-capacity nodes to advertise their true capacity, Gia clients assign tokens in proportion to the neighbors' capacities, rather than distributing them evenly between all neighbors. Thus, a node that advertises high capacity to handle incoming queries is in turn assigned more tokens for its own outgoing queries. The token assignment algorithm is based on Start-time Fair Queuing (SFQ) [43]. Each neighbor is assigned a fair-queuing weight equal to its capacity. Neighbors that are not using any of their assigned tokens are marked as inactive and the remained capacity is automatically redistributed proportionally between the remaining neighbors. As neighbors join and leave, the SFQ algorithm distributes its token allocation accordingly.

The *one-hop replication* is designed to improve the efficiency of the search process. Each Gia node actively maintains an index of the data content of each of its neighbors. These indices

are exchanged when neighbors establish connections to each other, and periodically updated with any incremental changes. Thus, when a node receives a query, it can respond not only with matches from its own content, but also provide matches from the content offered by all of its neighbors. When a neighbor is lost, either because it leaves the system, or due to topology adaptation, the index information for that neighbor gets cleared. This ensures that all index information remains mostly up-to-date and consistent throughout the lifetime of the node.

The *search protocol* is using a *biased* random walk. A Gia node selects the highest capacity neighbor for which it has flow-control tokens and sends the query to that neighbor. If it has no tokens from any neighbors, it queues the query until new tokens arrive. TTLs are used to limit the duration of the biased random walks. In addition to the TTL, query duration is limited by MAX RESPONSES. MAX RESPONSES represents the maximum number of matching answers that the query should search for Every time a node finds a matching response for a query, it decrements the MAX RESPONSES in the query. Once MAX RESPONSES hits zero, the query is discarded. Query responses are forwarded back to the originator along the reverse-path associated with the query. Since a node can generate a response either for its own files or for the files of one of its neighbors, the addresses of the nodes that own those files are appended to the forwarded query. This ensures that the query does not produce multiple redundant responses for the same instance of a file. A response is generated only if the node that owns the matching file is not already listed in the query message.

3.1.3 RAON Design

Based on the proposal in [37], RAON approach modifies Gia's design to address link instability and power constraints specifics to MANET. RAON tries to improve Gia's architecture by using a ranking system to categorize the neighbors according to their dynamic properties. The proposed solution is a Neighbor Coloring Scheme or NCS used by each node to keep track of the delay it experiences with each of its neighbors and the neighbors' energy levels. Each node will categorize its neighbors into three different classes depending on their dynamic attributes. The nodes are colored GREEN, YELLOW, and RED based on their ranks ranging from High, Medium, and Low. The idea is to make the biased random walk algorithm to also take neighbors' colors into consideration when making a forwarding decision in addition to the neighbors'

capacity levels. As in Gia, a RAON node requires a token in order to forward a query to a neighbor, and it only forwards to neighbors that it has not sent that query to before. Among the "available" neighbors, it groups them according to their color and selects the one with the greatest capacity from the highest color group [37].

The addition of NCS to the biased random walk in RAON has two scopes. In the first place, the algorithm tries to improve query search performance by avoiding unstable links. In the second place, NCS tries to increase the battery life of these neighbors by avoiding forwarding to low-energy nodes. Therefore, the rank of a neighbor is lowered if either the delay is high or the energy level is low. Each node can determine the delay by probing the neighbors periodically and measure the round trip time (RTT). The PROBE message can be a separate control message or piggybacked to another message. After a node sent out a probe message to its neighbors, it sleeps for *probe_interval* seconds and waits for an ACK message in the mean time. In case that it does not receive an ACK when the timer expires, it would consider the previous probe to be unsuccessful and assign a worst-case RTT to that neighbor. Each node monitors its energy level and determines its energy state relative to its battery size, and updates its neighbors only when there is a transition. Three energy states of HIGH, MEDIUM, and LOW are defined, which correspond to the three colors defined in NCS. Each energy state defined corresponds to different conditions in energy and latency level for each node. The flow control mechanism is also adapted to the MANET environment with the information provided by NCS by assigning tokens based on the neighbor's energy level and link delay.

The constant change in the physical connections due to node mobility is affecting the nodes' ability to select the optimal link in terms of stability. A mechanism proposed in RAON to address this problem is called Proactive Neighbor Replacement or PNR. RAON uses Proactive Neighbor Replacement mechanism to further improve the overall performance of NCS by changing the overlay topology adaptively. If a node calculates the latency to one of its neighbors to be high, it will try to connect to an overlay node that is not one of its neighbors and is reachable through a low latency MANET route. RAON suggests using the Ping-Pong method to discover new peers if the node is already connected to at least one peer. The main reason for replacing a neighbor is when the neighbor can affect the overall performance of a query search. Since the low energy level does not affect the existing query performance, it does not make a node candidate for neighbor replacement. However, dropping neighbor relationship with nodes

running at low energy level might help conserve their energy that may prevent future query miss. The main features of P2P file sharing applications are query search and file download. The operation of downloading a file consumes much more resources comparing to query search, since files are generally much larger than query messages in terms of size. When a node is running low in energy, it can start its power conservation strategies by disabling the download feature first. Also, if the node is running low in bandwidth due to congestion or serving multiple download requests, it can temporarily stop accepting download requests as well. After the download option is disabled, if the node is still suffering from a high consumption of resources, it can also make use of topology adaptation and flow control to reduce the consumption rate. By reducing the number of neighbors, it does not only reduce the number of queries it would receive, but also the number of update messages. This approach is targeted for conserving energy rather than bandwidth. When selecting which neighbor to drop, it should choose the ones that are colored as RED first, especially the ones that are low in power [37].

3.2 CAN P2P System

The Content-Addressable Network or CAN [11] is a distributed decentralized P2P infrastructure based on DHT. CAN is designed to be scalable, fault-tolerant, and self-organizing. The architecture is based on a logical multi-dimensional Cartesian coordinate space on a multi-torus. This *d*-dimensional coordinate space is completely logical without any relation with physical architecture. The entire coordinate space is dynamically partitioned among all the nodes in the system such that every node controls its individual, distinct zone within the overall space. A CAN peer maintains a routing table that holds the IP address and coordinate information of each of its neighbors in the coordinate space. Using the neighbor coordinates, a peer routes a message toward its destination using a greedy forwarding algorithm to the neighbor peer that is closest to the destination coordinates. As shown in Figure 3.2, the logical coordinate space is used to store {**key, value**} pairs: to store a pair {**K,V**}, **key K** is deterministically mapped onto a point P in the coordinate space using a uniform hash function.

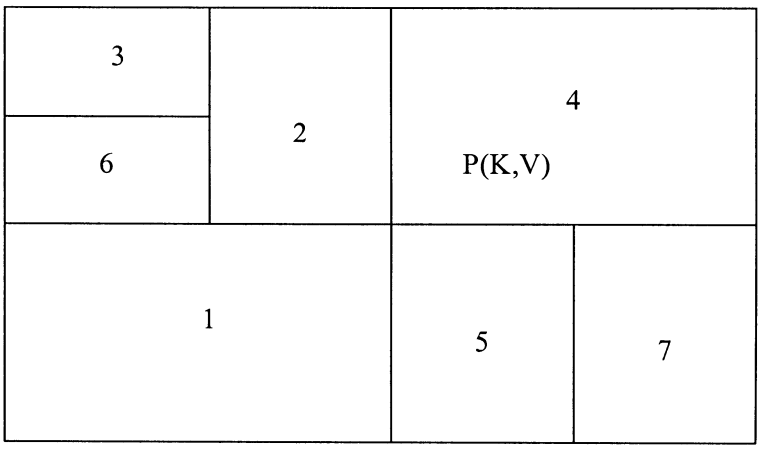

Figure 3.2 2-dimensional CAN coordinates space

The corresponding (key, value) pair is stored at the node that owns the zone within which the point P lies. To retrieve an entry corresponding to key K, any node can apply the same deterministic hash function to map K onto point P and then retrieve the corresponding value from the point P. If the point P is not owned by the requesting node or its immediate neighbors, the request must be routed through the CAN infrastructure until it reaches the node in whose zone P lies. A node learns and maintains the IP addresses of those nodes that hold coordinate zones adjoining its own zone. This set of immediate neighbors in the coordinate space serves as a coordinate routing table that enables routing between arbitrary points in this space. Routing in CAN is an essential component for a proper functioning of the system. Routing in a Content Addressable Network works by following the direct line path through the Cartesian space from source to destination coordinates. Because multiple paths can exist between two points in the space, even if one or more of a node's neighbors, a node can automatically route along the next best available path. If however, a node loses all its neighbors in a certain direction, and also the repair mechanisms fails, then greedy forwarding may temporarily fail. In this case, a node may use an expanding ring search to locate a node that is closer to the destination than itself. The message is then forwarded to this closer node, from which greedy forwarding is resumed [11].

The process of joining the system of a new node requires a bootstrap mechanism. The bootstrap mechanism similar to [31] will assume that a node joining the network will access a available DNS server associated with a CAN domain name to retrieve the IP addresses of randomly chosen peers existent in the network. The new node then randomly chooses a point P in the space and sends a JOIN request destined for point P. This message is sent into the CAN

via any existing CAN node. Each CAN node then uses the CAN routing mechanism to forward the message, until it reaches the node in whose area P lies. The node that owns the area where P is located splits its zone in half and assigns one half to the new node. In a 2-dimensional space, a zone would first be split along the *X* dimension, then *Y*, and the process continues in the same way. The {**K,V**} pairs from the half zone to be handed over are also transferred to the new peer. After obtaining its zone, the new node learns the IP addresses of its neighbor set from the previous occupant. This set is a subset of the previous occupant's neighbors, plus the previous neighbor itself. Similarly, the previous occupant updates its neighbor set to eliminate those nodes that are no longer neighbors. After space splitting, both the new and old nodes' neighbors must be informed of this reallocation of space. Every node in the system sends an immediate UPDATE message, followed by periodic refreshes, with its currently assigned zone to all its neighbors. These updates ensure that all of their neighbors will quickly learn about the change and will update their own neighbor sets accordingly. Figure 3.3 show an example of a new node called A joining a 2-dimensional CAN. Based on the algorithm described above, the node B will split its space and the information from B is handed over to A. The number of neighbors a node maintains depends on the size of the coordinate space and is independent of the total number of nodes in the system.

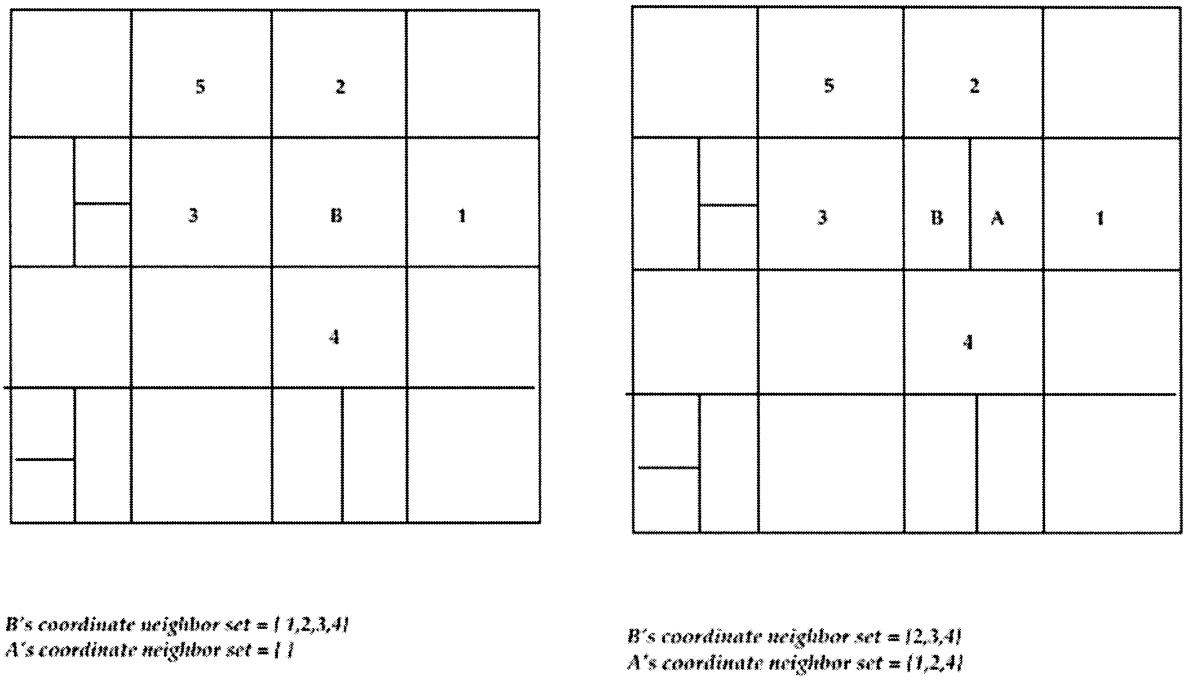

Figure 3.3 Joining process in CAN

When nodes leave a CAN, an immediate takeover algorithm ensures that one of the failed peer's neighbors takes over the zone. The peer updates its neighbor set to eliminate those peers that are no longer its neighbors. Every peer in the system then sends soft state updates to ensure that all of their neighbors will learn about the change and update their own neighbor sets. If the zone of one of the neighbors can be merged with the departing node's zone to produce a valid single zone, then this is done. If not, then the zone is handed to the neighbor whose current zone is smallest, and that node will then temporarily handle both zones. Under normal conditions a node sends periodic update messages to each of its neighbors giving its zone coordinates and a list of its neighbors and their zone coordinates. The prolonged absence of an update message from a neighbor signals its failure. Once a node has decided that its neighbor has died it initiates the takeover mechanism and starts a takeover timer running. When the timer expires, a node sends a TAKEOVER message to all of the failed node's neighbors. On receipt of a TAKEOVER message, a node cancels its own timer if the zone volume in the message is smaller that its own zone volume, or it replies with its own TAKEOVER message. In this way, a neighbor is efficiently chosen that is still alive and has a small zone volume.

Improvement of the CAN algorithm can be accomplished by maintaining multiple, independent coordinate spaces, with each peer in the system being assigned a different zone in each coordinate space, called *reality*. For a CAN with multiple realities, a single peer is assigned multiple coordinate zones, one on each reality available, and this peer holds multiple independent neighbor sets. The contents of the hash table are replicated on every reality, thus improving data availability. For data availability improvement, CAN also uses different hash functions to map a given key onto multiple points in the coordinate space. This results in the replication of a single {key, value} pair at distinct peers in the system. A {key, value} pair is then unavailable only when all the all pair replicas are simultaneously unavailable. Thus, queries for a particular hash table entry could be forwarded to all peers in parallel, thereby reducing the average query latency, and reliability and fault resiliency properties are enhanced.

Chapter 4 Evaluation

We used NS-2 simulator to implement both RAON and CAN and evaluate their relative performance. We used RAON implementation in NS-2 as reported in [37], and implemented CAN in the same code base. The simulation environment will be described in the next section of the chapter. Based on the simulations results a comparison between the two systems is presented in the last section of the chapter.

4.1 Simulation Environment

CAN and RAON implementations were simulated with the Network Simulator NS-2 [35]. NS-2 is an open source simulator developed by the VINT project at the University of California at Berkeley, with the mobile and wireless extensions contributed by the MONARCH research group of Carnegie-Mellon University. The TCP socket and peer-to-peer extension developed by the Networking and Telecomm Group at Georgia Institute of Technology was also installed. The simulations are based on NS-2 version 2.26 installed on Debian Linux 3.1 Sarge. The IEEE 802.11 standard MAC layer is used at the physical level. Each node is equipped with an antenna that is 1.5m above the ground. The two-way ground propagation model is used with a transmission power of 281.8mW, which results in a transmission range of approximately 250m. The energy model provided by NS-2 is used to model the node's energy consumption and all nodes start with the same initial energy level. Node movements are modeled with the random waypoint mobility model, where each node chooses a destination randomly within the simulation area using a uniform random speed between 0 and *max_speed*. When the node reaches the destination, it pauses for a fixed amount of *pause_time*, which is set to 30 seconds in all the scenarios, and then move on to another destination. The simulation objective is to study the RAON and CAN behavior in similar conditions. The evaluation of their performances under different system configurations and network conditions will show the major characteristics of these two algorithms. RAON implements simple Gia design without flow control and one hop replication. RAON algorithms used are NCS and PNR. In case of CAN, the implementation is based on a 2-dimension space. Table 4.1 shows the network level simulation parameters.

Parameters	Values
Simulated Period *sim_time*	1000 sec
Number of Mobile Nodes	100
Simulation Area	500 m x 500 m
Maximum Speed *max_speed*	2m/s, 5m/s, 20m/s
Pause Time *pause_time*	30 sec

Table 4.1 Network simulation parameters

The speeds of 2m/s, 5m/s, and 20m/s are used to model the movements of pedestrians, non-motorized vehicles, and motor vehicles respectively. For node density, we keep the number of mobile nodes constant, and use a simulation area of 500m x 500m. Thus, we generated three scenarios for all combinations of speed and dimension. 2:500x500, 5:500x500, 20:500x500 represents the simulation scenarios.

The simulation begins with a defined MANET topology where the nodes are randomly placed over the simulation area. AODV is the underlying routing protocol in MANET [8]. AODV was chosen because it is an on-demand routing protocol with local route repair, which shows a good performance among different protocols [36]. At the P2P level, each overlay node uses a uniform random number between [0, *sim_time*/2] to determine when to join the network. Therefore, no P2P topology is defined in the beginning of the simulation. A virtual bootstrap server exists in the simulation to provide the existing P2P nodes in the network. After a node is connected to the overlay network, it periodically exchanges with its neighbors update messages every *update_interval*. The *update_interval* was set to be 30 seconds. Queries are generated at a minimum rate of 1 query per 10 seconds to model the behavior of aggressive P2P users. Each query contains a keyword, which is represented by an integer between [0, 499], and every keyword is mapped to a set of files. The files located on each node are generated randomly, but the number of files and the file sizes are dependent on the node's capacity level. CAN generates the files for the whole space based on two hash functions. The process that generates files in CAN is completed in the bootstrap process for all nodes. The hash functions used by CAN generate X and Y coordinates for the point in space where the object will be located. The searching process will use a keyword as a search object. Based on the same hash functions used

for file generation, two coordinates X and Y will be created hashing this keyword. The coordinates created are similar with the ones created when a file was generated initially. A node that owns the file name matching the searched keyword will also check that the coordinates associated with that keyword will be located within the coordinate space ranges owned by it. If the result is true, CAN considers as the query hit. In RAON the keyword searching will be based on simple matching between the file owned by a node and the searched keyword. Table 4.2 summarizes all the simulation parameters used at the P2P level.

filen = is the file name generated randomly

Parameters	Values
Number of peers	10, 20, 30, 40, 50
Start Time	Uniform random between [0, 500]
Query Generation Interval	Uniform random between [0, 10]
Number of files stored at each peer	Uniform random between [0, 500]
update_interval	30 sec
query_timeout	120 sec
Latency thresholds TH1, TH2 for RAON	1 sec, 3 sec
Energy thresholds TH1, TH2 for RAON	50 %, 20%
max_responses	1
TTL	32
Hash function for X coord. in CAN	F(x)= (2*filen^2 + filen%500)%500
Hash function for Y coord. in CAN	F(y)= (256*filen + filen%500)%500

Table 4.2 P2P simulation parameters

4.2. Simulation Results

RAON uses biased random walk and NCS scheme to forward queries. CAN forwards queries based on the Cartesian distance. When a node starts searching over the overlay network, the node checks to see if any of its neighbors is situated closer to the target location. The algorithm is based on calculating the minimum Cartesian distance. First, a node sending a query determines the minimum distance from its neighbors' area to the point that a query has to reach. Once this minimum distance is determined, the node generating the search sends the query to the neighbor considered closest to the destination point. The searching process continues for each

node until the coordinate points where the object is located is reached. The forwarding algorithm for CAN does not take into account any information related to link stability or neighbor's capacity, as is the case for RAON. The main performance metrics used for evaluation of forwarding process are query search rates and query delays. A query is considered successful if the query originator receives at least one query hit response before the query timed out, and the query delay is the time it takes for the originator to receive the query hit response.

We chose to use the RAON variant with NCS and PNR to see the full functionality of these two improvements to Gia. CAN is implemented as a 2-dimension coordinate space with its algorithm for routing and searching. In our 2-dimensional implementation simulation, each node will check to see which neighbor is the closest to the point in space where the searched object is located based on minimum Cartesian distance.

In Figure 4.1 the results for RAON show the query success rate.

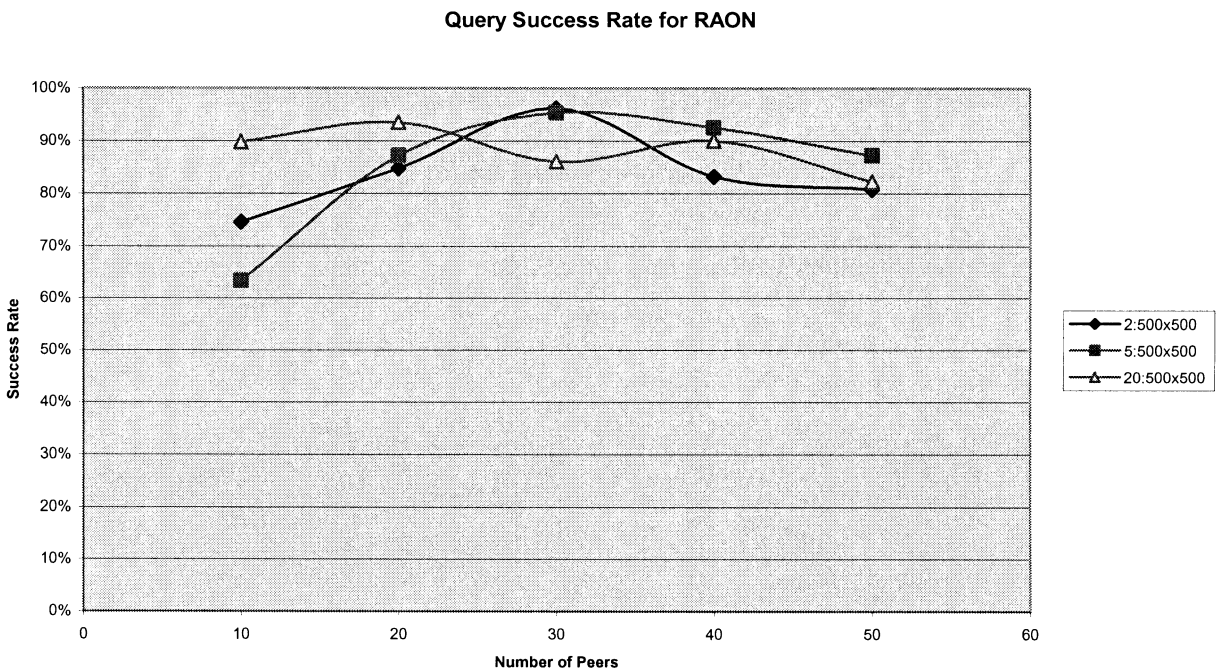

Figure 4.1 Query Success Rate for RAON.

Three different scenarios for 2m/s, 5m/s and 20m/s are evaluated with a number of peers varying from 10 to 50. The success rate in case of RAON remains at a high level, between 60% and 90%, for different number of peers. The variation in RAON is very low for query success rate even with increasing the number of peers. These results can be mainly due to the quality of RAON's forwarding decision and its well-replicated content over the network.

In Figure 4.2 the results for CAN show the query success rate. The CAN behavior shows a very low rate of query success rate. By increasing the peer number the drop in the query success rate is substantial.

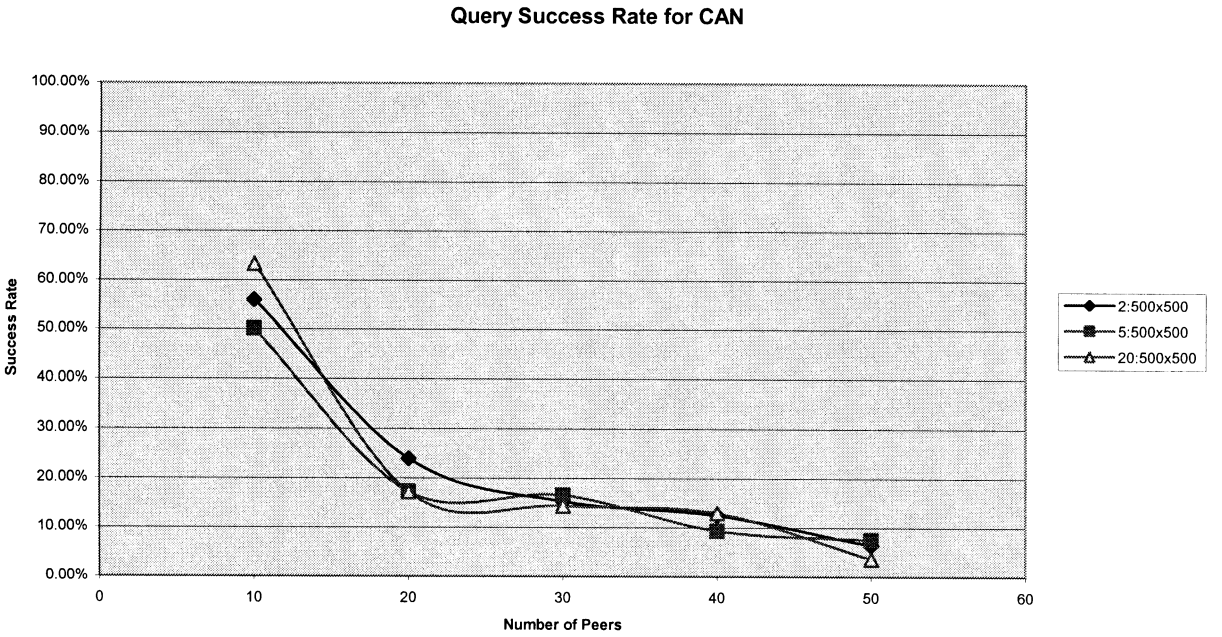

Figure 4.2 Query Success Rate for CAN

Figure 4.3 shows the query success rate for CAN in the moment when one node failed.

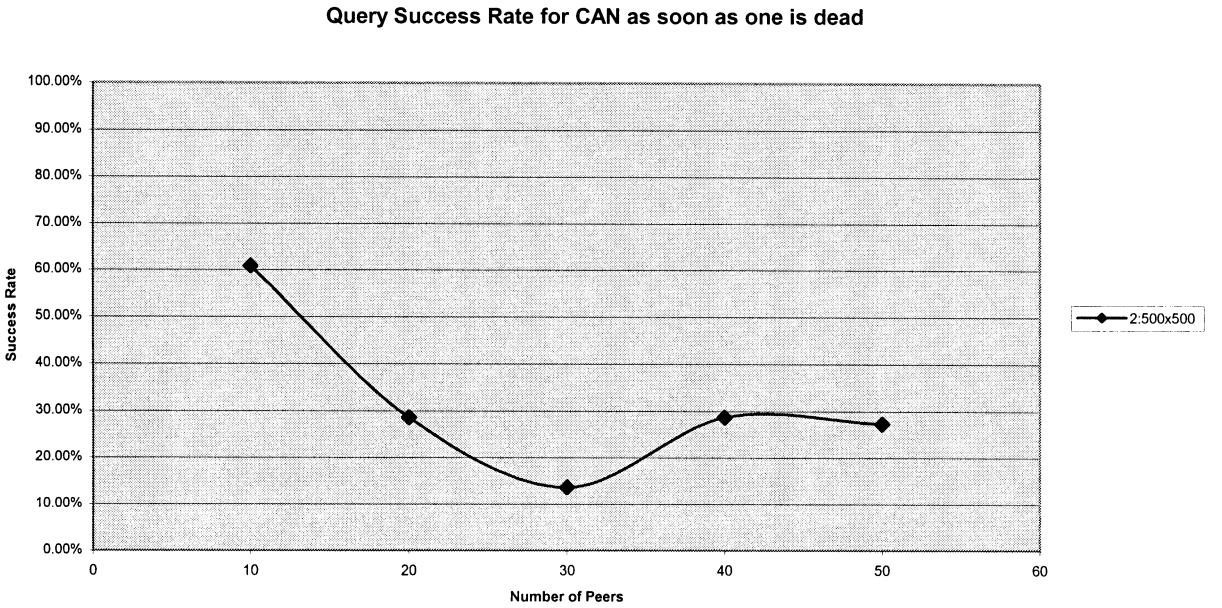

Figure 4.3 Query Success Rate for CAN as soon as one node is dead

RAON query delay values showed in Figure 4.4 present a slow increase with increasing the number of peers. It is obvious from the figure that the query success rate is not affected by the increase in query delay. Thus, RAON is able to maintain high query success rate.

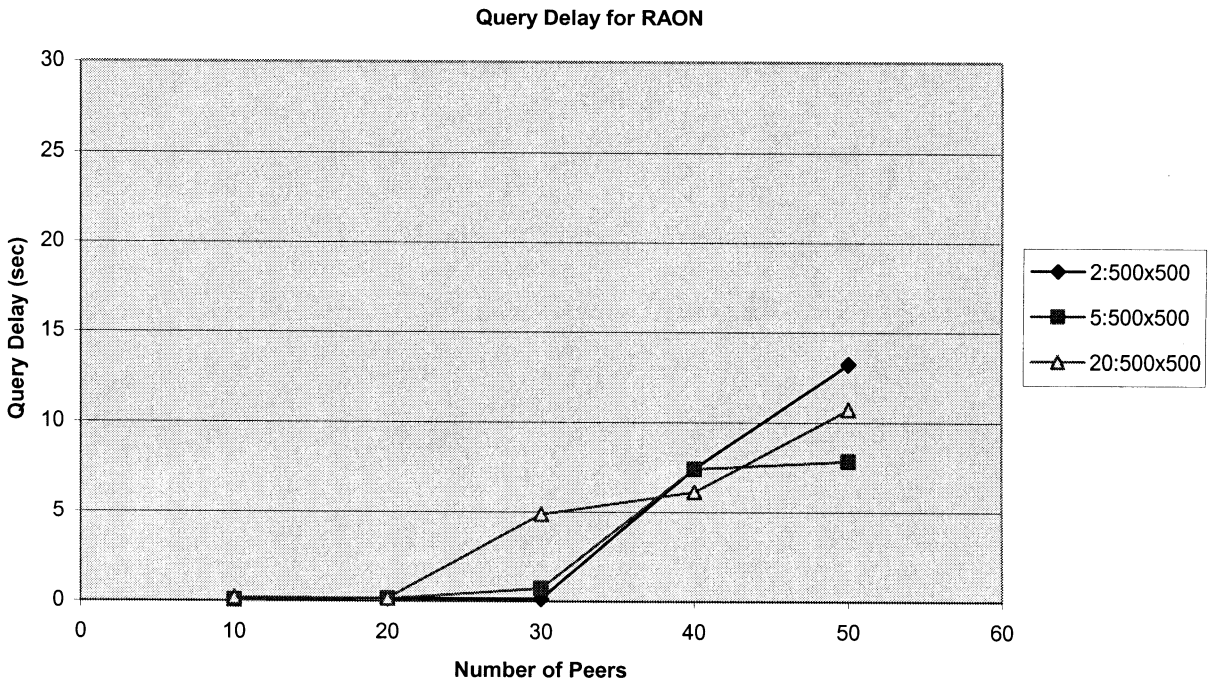

Figure 4.4 Average Query Delay for RAON

A reason for poor query success rate in CAN, as we can see in the Figure 4.5 for average query delay, can be a very high level of delay exhibited. For a small number of peers, in our case ten peers, CAN show a low query delay value and the query success rate presents an acceptable level. By increasing the number of peers, the average query delay values will increase and consequently query success rate drops.

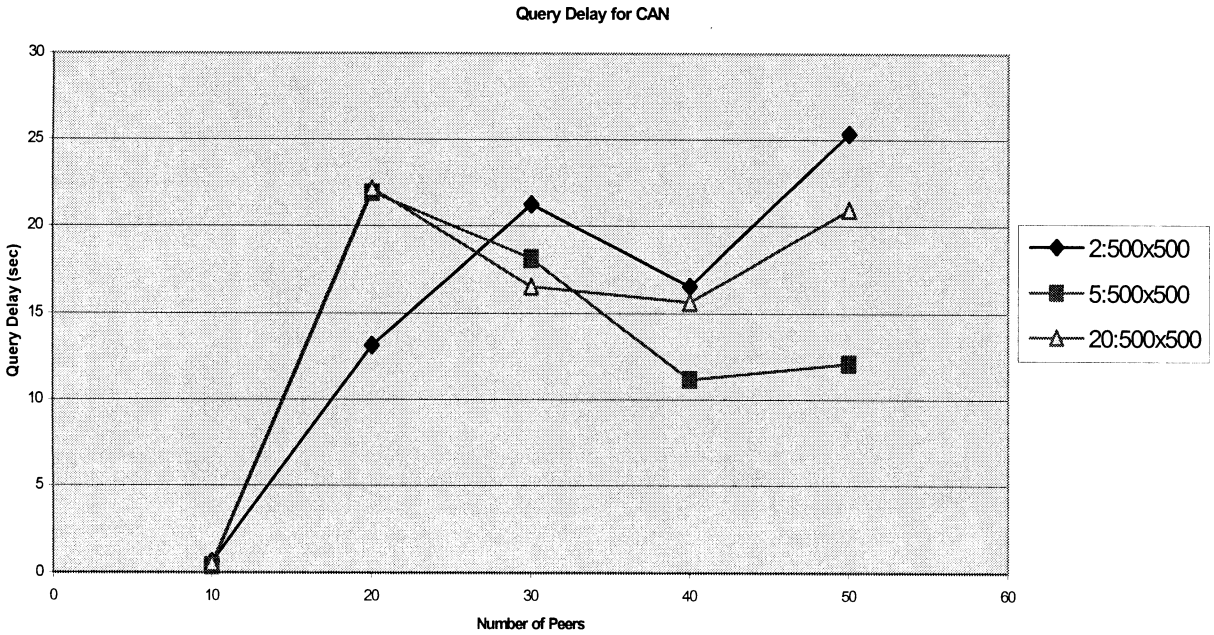

Figure 4.5 Average Query Delay for CAN

In Figures 4.6, 4.7 the maximum query delays for both systems, RAON and CAN, are presented. The maximum query delay for RAON shows increasing values for higher number of peers. Compared with RAON, CAN is largely affected by a very high number of query delays with values above the query_timeout value, which is 120 seconds. In this case, CAN suffers a performance hit because the queries experiencing high delays are counted as misses. The query success rate for CAN is negatively affected by a large number of high delay queries considered as misses.

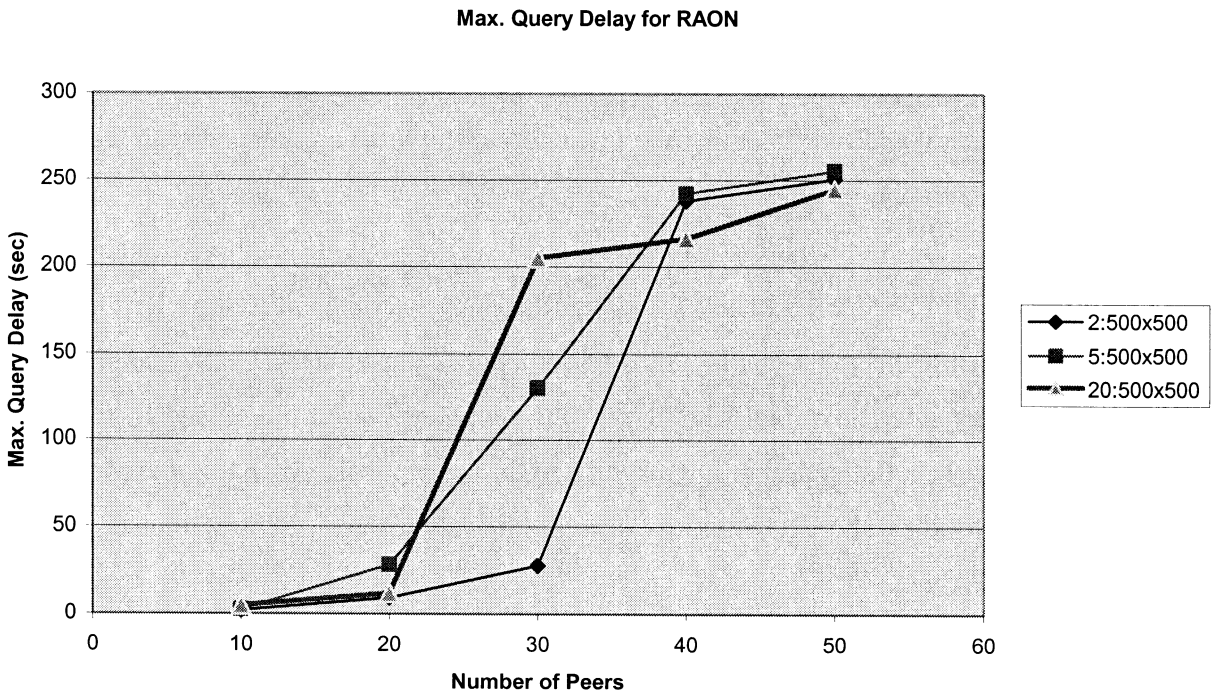

Figure 4.6 Maximum Query Delay for RAON

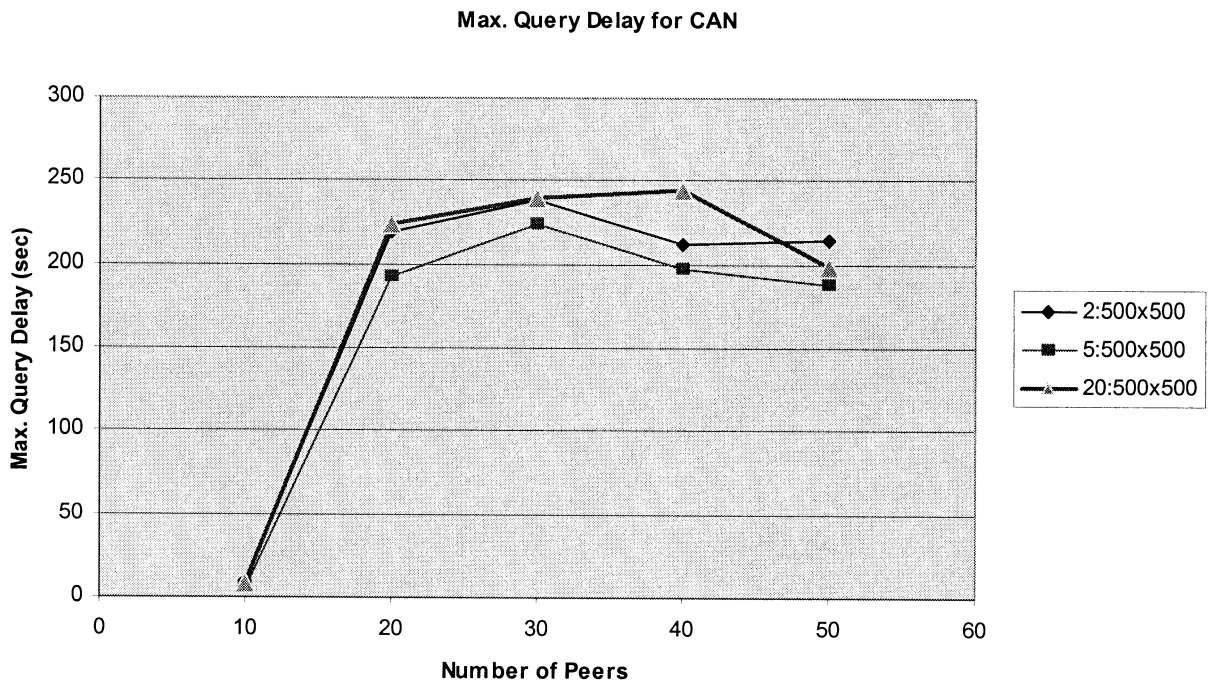

Figure 4.7 Maximum Query Delay for CAN

RAON uses biased random walk to forward queries. The link color in RAON is expected to improve the quality of forwarding decision and as a consequence to improve the query success

rates and query delays. In order to compare the quality of forwarding decisions, the statistics for using the colored links for both systems, RAON and CAN, are calculated. The link colors are presented in different graphs separately for forward and backward paths. To distinguish the red links that are high delay and red links that are low energy, the links are labeled with Red (delay) and Red (ener) respectively.

In Figure 4.8 the number of green links used by RAON is far higher than the number used by CAN shown in Figure 4.9. The forwarding decision mechanism used by RAON plays a major role in choosing the most stable link in the network. In case of CAN, the forwarding decision does not consider link stability, therefore the number of red links used in forwarding process is increased. It is noticed, that in case of RAON the number of red links used for query forwarding is negligible, showing that its forwarding algorithm performs better than the algorithm used by CAN.

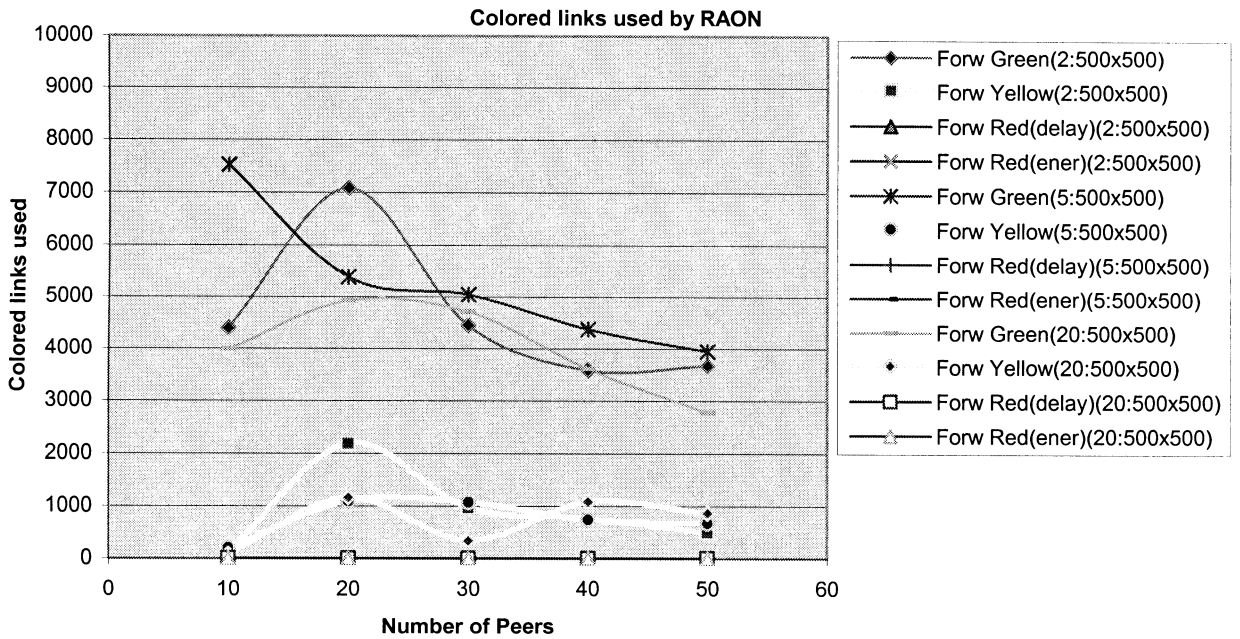

Figure 4.8 Colored links used by RAON

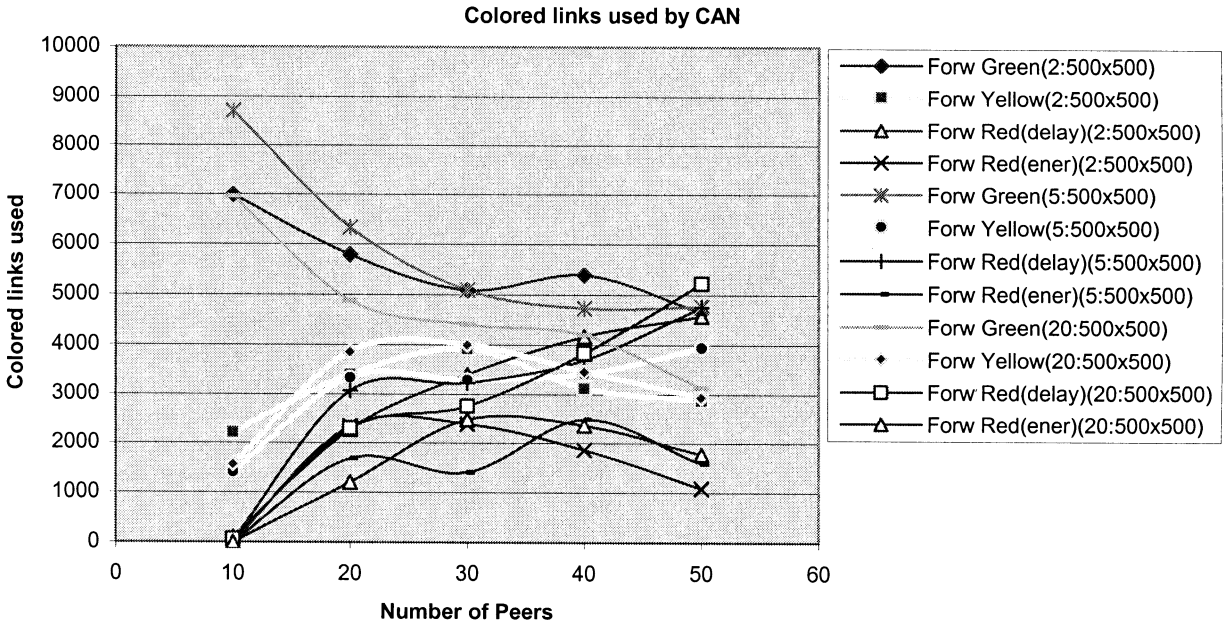

Figure 4.9 Colored links used by CAN

Figure 4.10 illustrates colored links used in backward paths by RAON. RAON uses more red links in the backward path carrying responses back to the query originator. The number of red links has increased as the numbers of peers are increased.

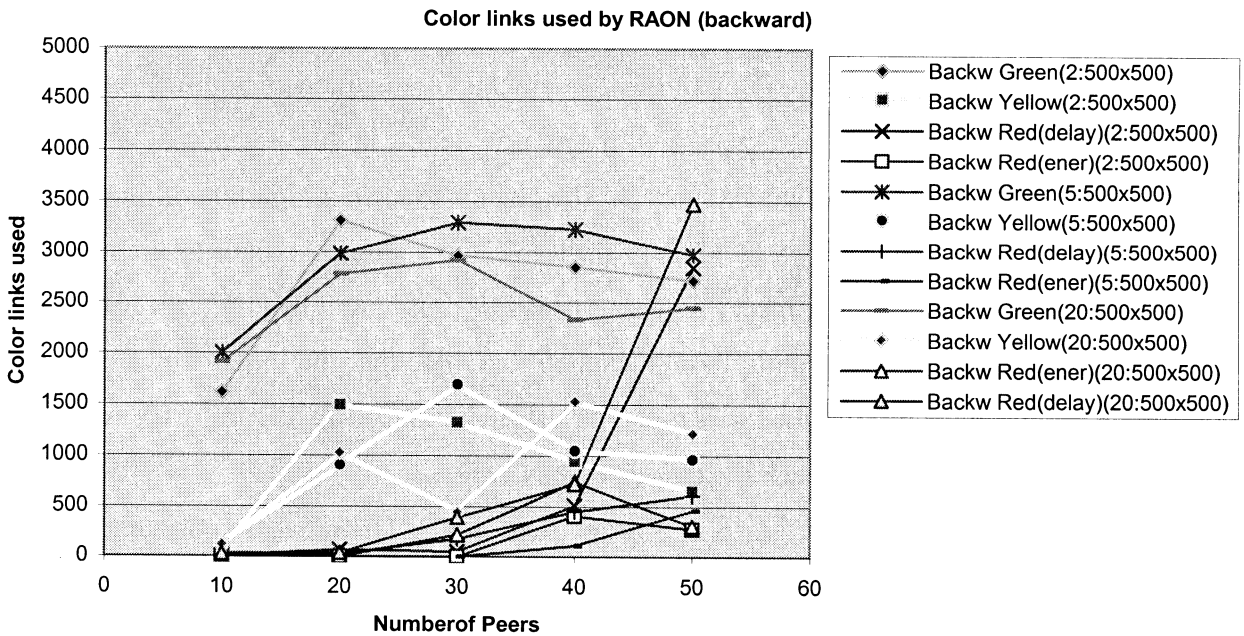

Figure 4.10 Colored links used by RAON (backward)

For CAN, in Figure 4.11, the backward red links also increase with the increasing number of peers. In contrast with RAON, CAN responses traverse more red links along the backward paths.

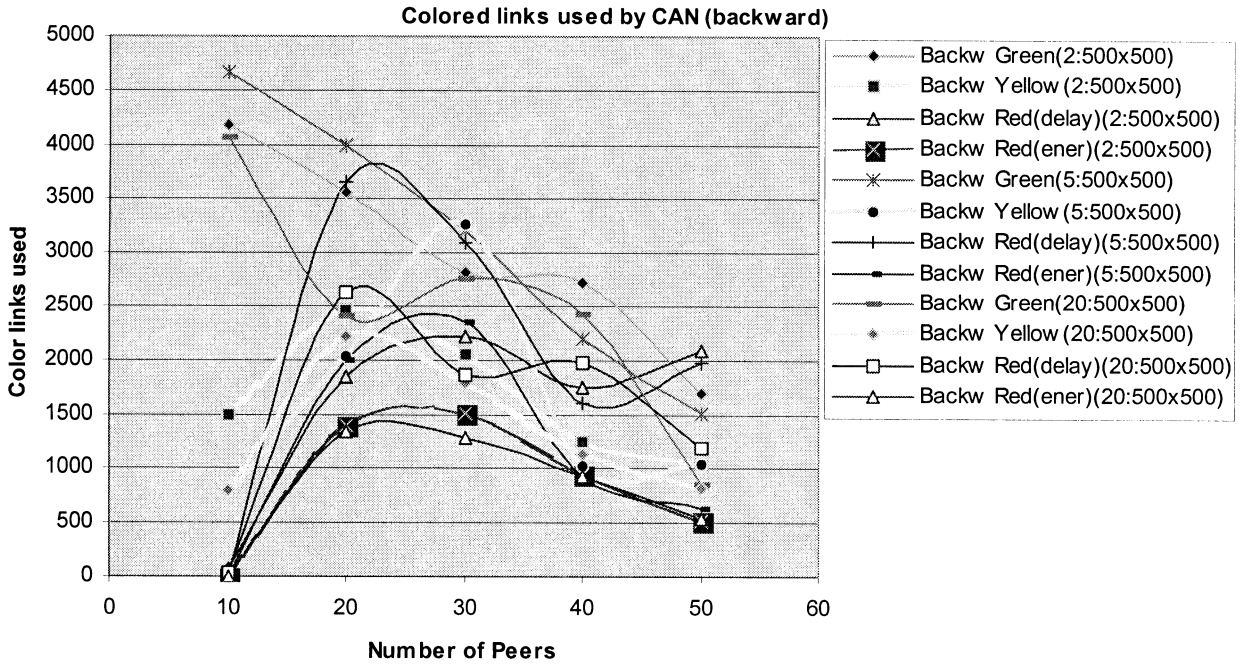

Figure 4.11 Colored links used by CAN (backward)

The large numbers of green links utilized by RAON is a result of the forwarding mechanism used. Thus a source does not generate queries unless it finds a good link, which can be green or yellow. Even if finding a good link may cause delays in query generation time, this method ensures a better chance of query success. The chance to have a high rate of query hits is based on the fact that a node forwards a query on stable links along the path to the next hop. And, the next hop can be able to process a query instead of dropping it as long as it has enough power to do it. On the backward path, RAON cannot guarantee that the initial forwarding path chosen as a stable link will remain the same. The effect is noticed in Figure 4.10, where the number of red links used to send back the query hits has increased.

The number of good links, green or yellow used by CAN, decreases consistently as the number of peers has increased. The color for selected links in case of query forwarding has little importance in case of CAN. CAN does not consider the stability of the link in forwarding process as it is noticed in the large number of red links selected during this process. The red links

have a high probability to lead to link failure. Once a node along the route detects a link failure, it starts a local repair process. During the local repair process, packets to be sent to destination are queued at the node executing the local repair. The delays of the queued packets or the lost packets over the failed links force the source node for the respective packets to generate retransmissions at the TCP level. Also, dropped packets because of the full queues cause retransmissions. As is seen in Figure 4.12, CAN system has to deal with situations where a very high number of retransmissions occur.

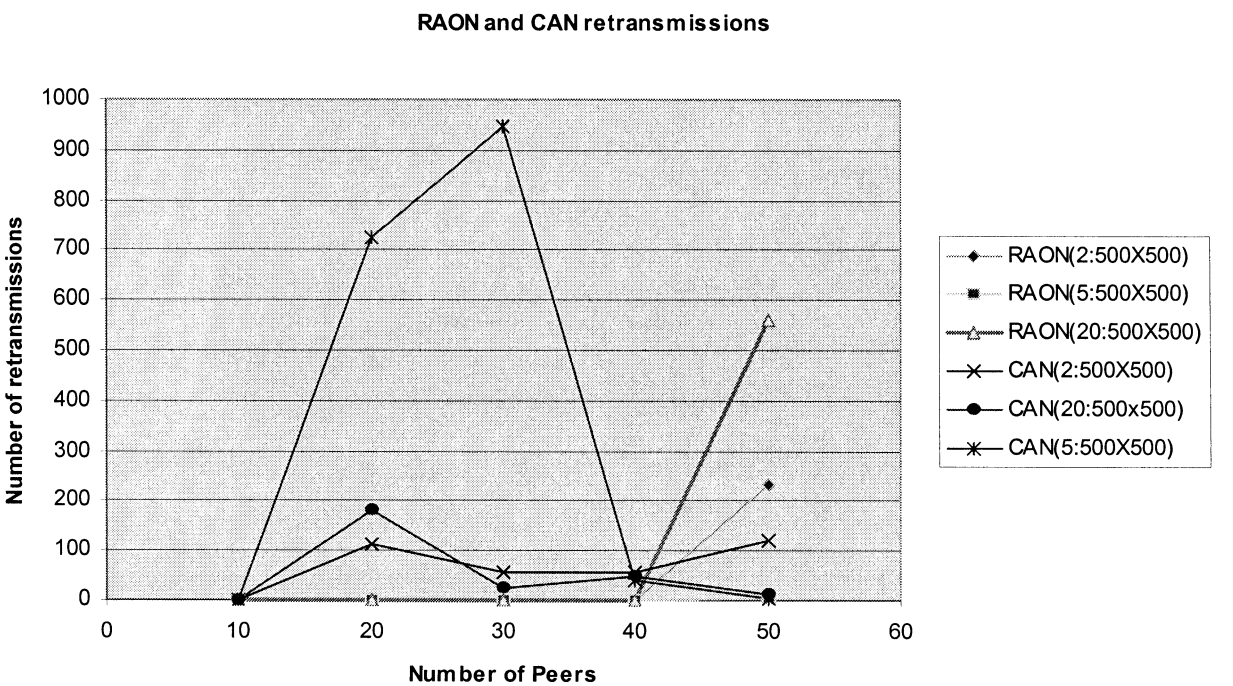

Figure 4.12 Number of retransmissions

RAON and CAN present different RTT values as is shown in Figures 4.13 and 4.14. The RTT values have an increasing trend for both systems by increasing the number of peers. For CAN, the RTT growth is more rapid and appears even for low number of peers. RAON considers RTT values in its forwarding decisions. Depending on these values, RAON is able to use stable links and to sustain a good quality of query success rate for high number of peers.

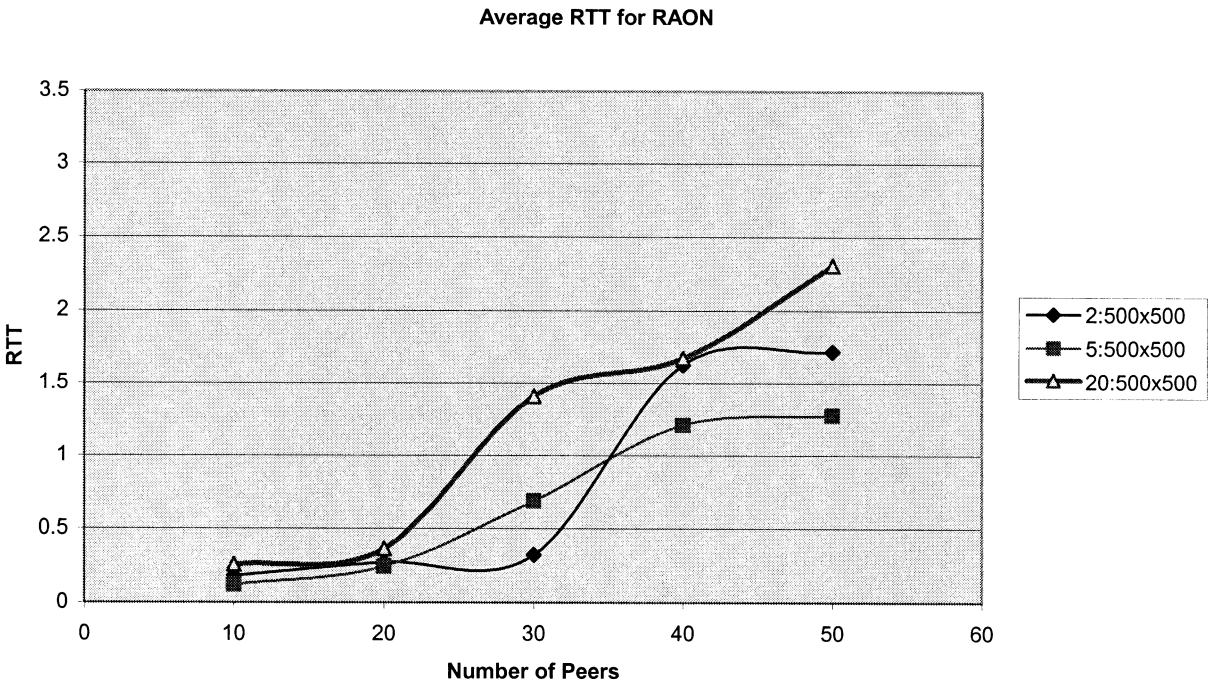

Figure 4.13 RTT for RAON

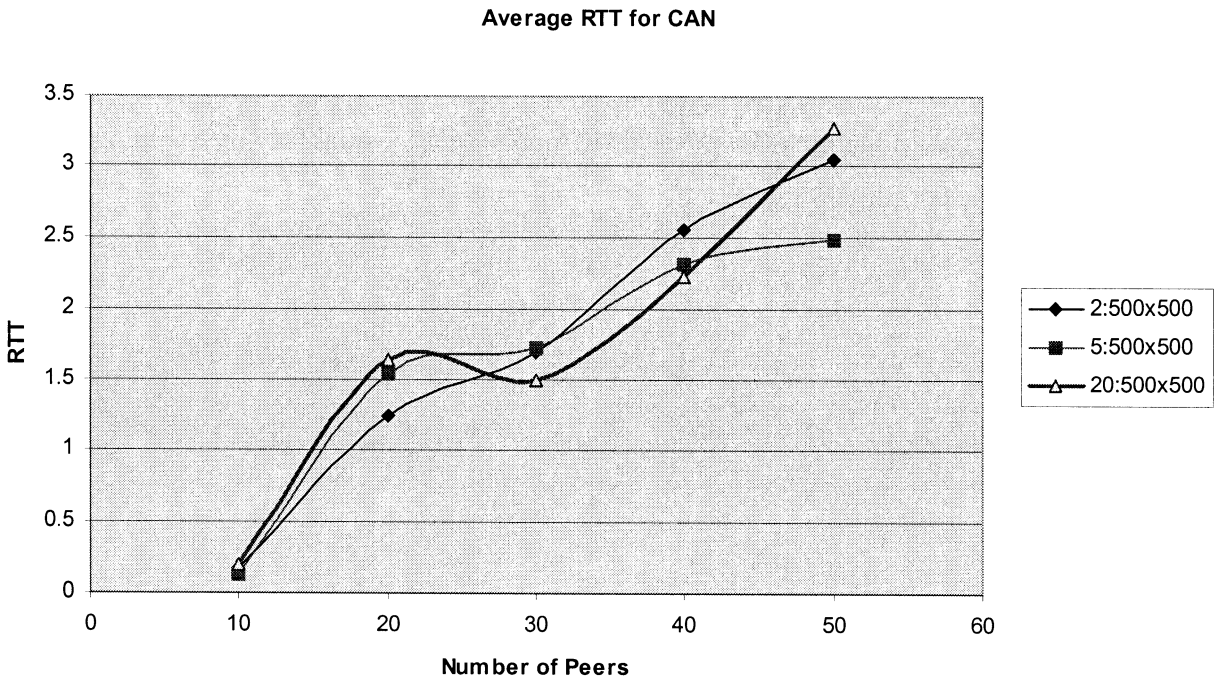

Figure 4.14 RTT for CAN

The CAN implementation used for simulations does not implement the modifications to improve its performance as described in [11]. The multi-dimensional coordinate space and multiple realities address the latency issue. Thus, by increasing the number of dimensions, a node will have more neighbors and more choices in selecting the nodes that are closer in terms of latency. Also, the multiple realities allow a node to check its neighbors on each reality and forwards the message to the neighbor with coordinates closest to the destination point. Another possible improvement might come by sending queries in parallel to multiple nodes that own replicated (key, values) pairs thereby reducing the average query latency. This could be achieved by using multiple hash functions. These multiple hash functions have the role of mapping a single key to multiple points in the coordinate space and replicating a single (key, value) pair at multiple distinct nodes in the system [11].

The overlay hop count, which is the number of overlay hops that the query traverses before finding a match, is also an important performance measure that can be useful for our comparison of RAON and CAN. The simulations showed for RAON that the average overlay hop counts is only a single hop. This also shows that the files are well-replicated over the network, since most queries are able to find a match with only one hop. In case of CAN, the

replication is not exploited during file search. We added in our CAN implementation the uniform partitioning concept where a node joining the network splits the space with an existent node with the largest available space. A uniform partitioning of the space is desirable to achieve load balancing. Multiple hash functions can be used to improve data availability by mapping a key on multiple points in the coordinate space.

In Figure 4.15, the average query hop count for CAN is shown for different speeds.

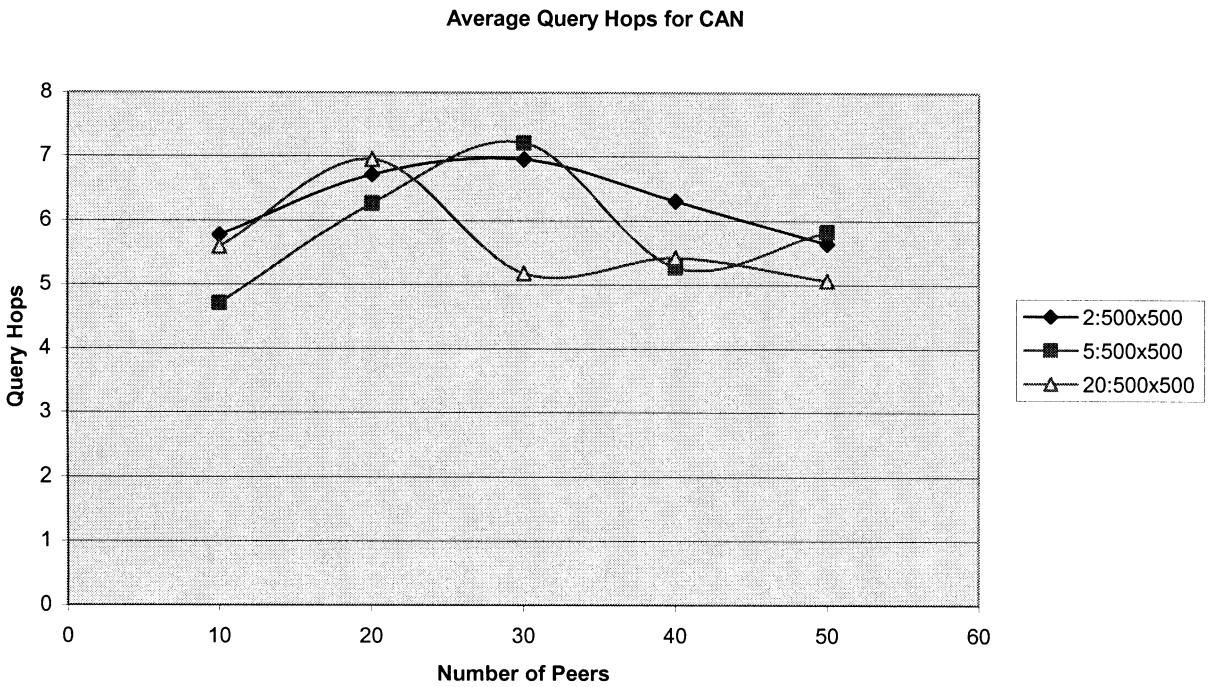

Figure 4.15 Average Query Hops for CAN

In case of RAON overlay hop count is not considered because of its low value. For CAN, the number of query hops is high for increasing number of peers. These results give us an indication of algorithm's slow performance in forwarding decision based only on minimum distance to the target point. In contrast, RAON needs only one hop to reach its target. CAN nodes have no information about the stability of a link or neighbor capacity to which forward a query.

In Figure 4.16 the average queryhit hop number for CAN is shown. It is observed a slightly decreasing in number of hops for a larger number of peers in the system.

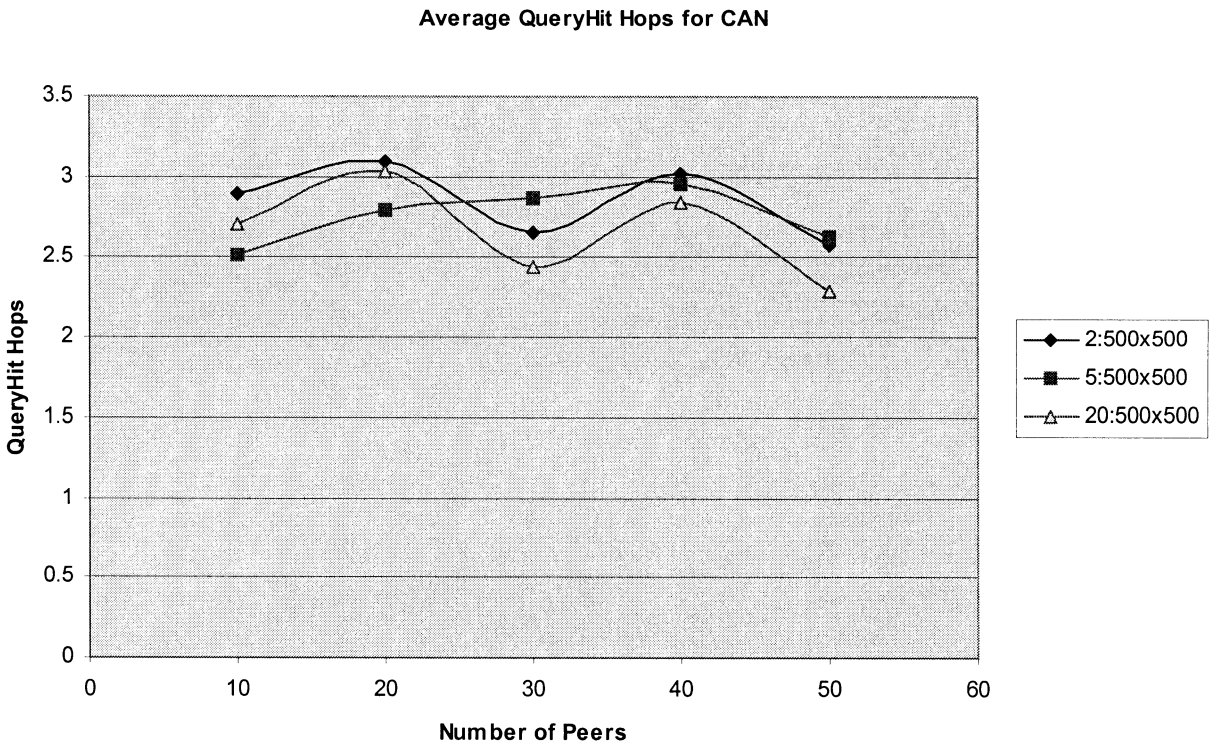

Figure 4.16 Average QueryHit Hops for CAN

The physical hop counts shown in Figures 4.17, 4.18 reflect the neighbor relationship. The average hop count between two neighbors for both cases, RAON and CAN, shows that this value is not dependent on the number of peers or speed.

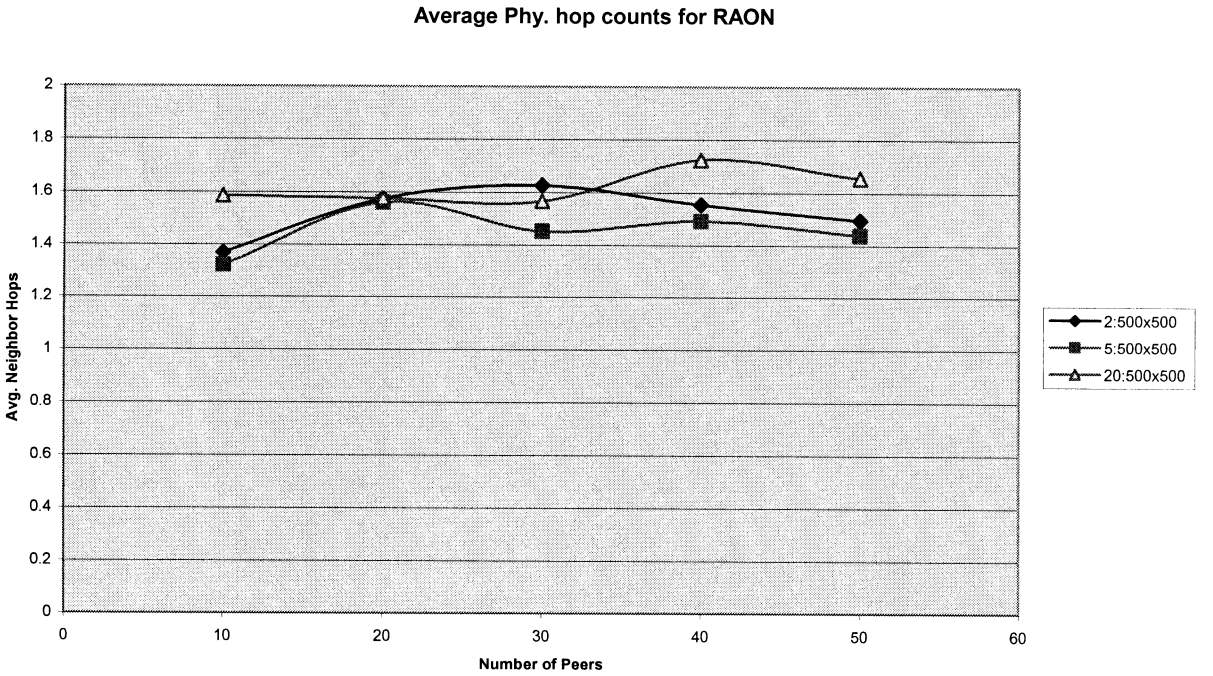

Figure 4.17 Average physical hop counts between the neighbors for RAON

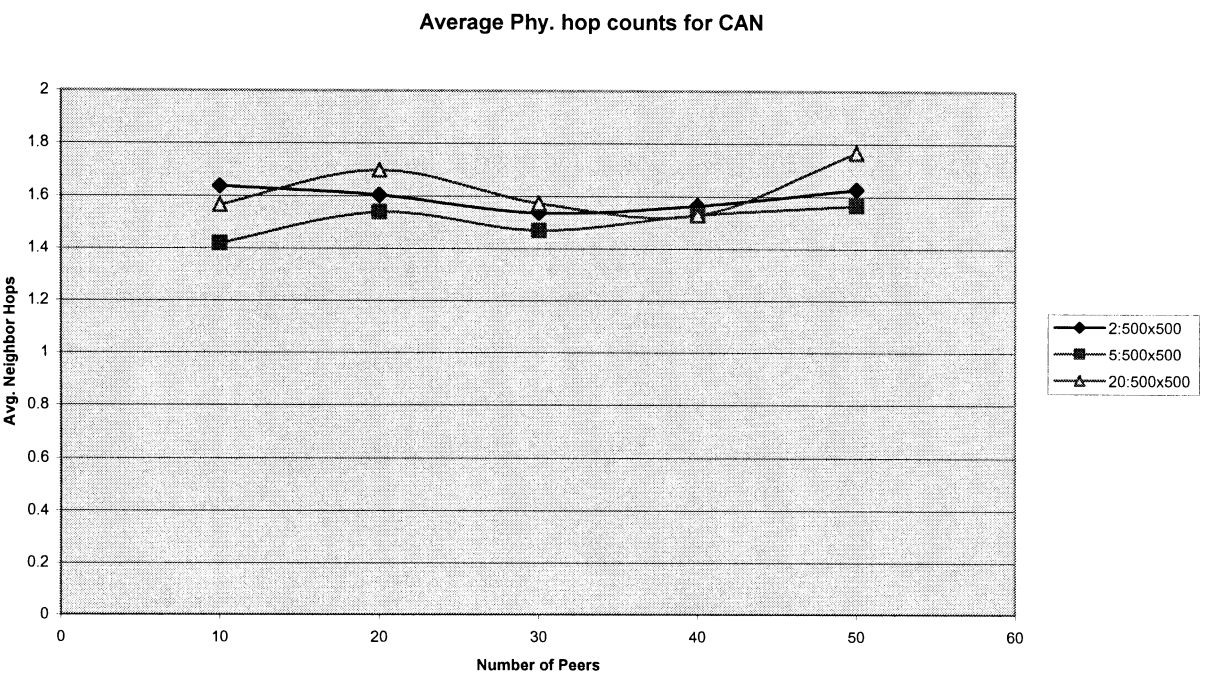

Figure 4.18 Average physical hop counts between the neighbors for CAN

For the worst-case scenarios, presented in Figures 4.19, 4.20, the maximum number of hops between two neighbors shows that some of the neighbors are 8 or 10 hops away from each other. For the connections that involve too many physical hops, more bandwidth and energy of the network is consumed and also the chance of encountering link failures is increased.

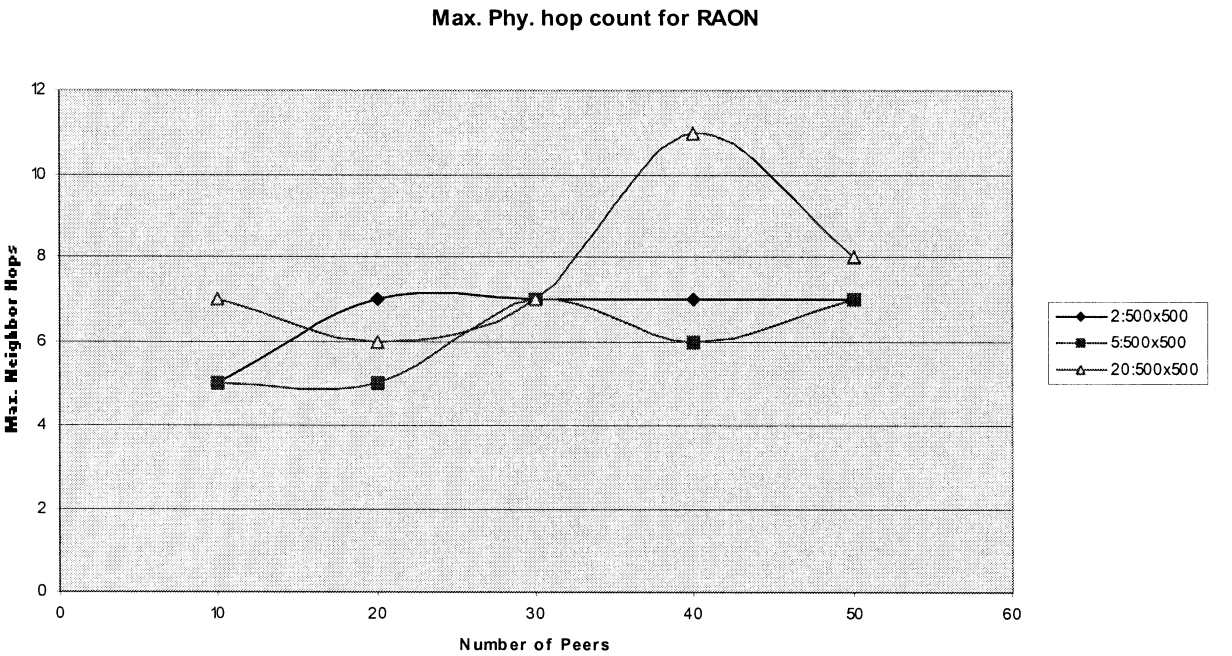

Figure 4.19 Maximum physical hop counts between the neighbors for RAON

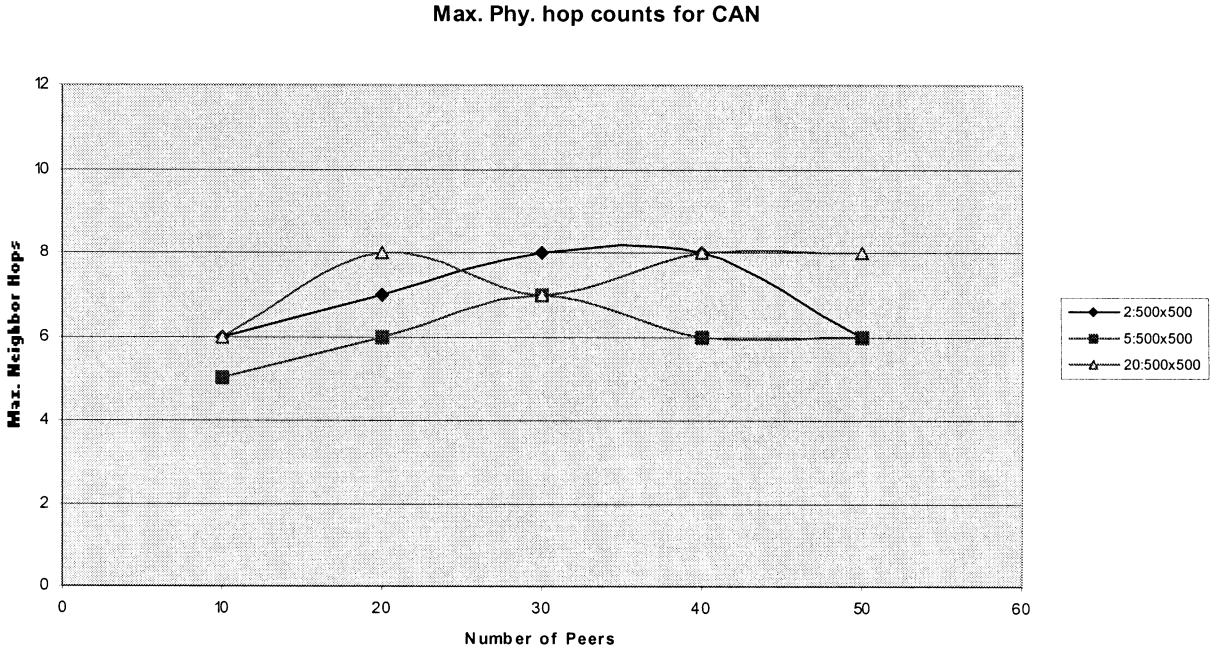

Figure 4.20 Maximum physical hop counts between the neighbors for CAN

In mobile networks to conserve energy consumption is an important design objective. In our simulations this is measured by keeping the statistics of the number of dead nodes and dead time. A dead node is a node that runs out of power and failed before the simulation ends. Dead time represents the average time for a node to run out of power. Figures 4.21, 4.22 show the number of dead nodes. Figure 4.23 shows dead time for RAON. The effect of energy consumption in RAON is observed in case of high number of peers.

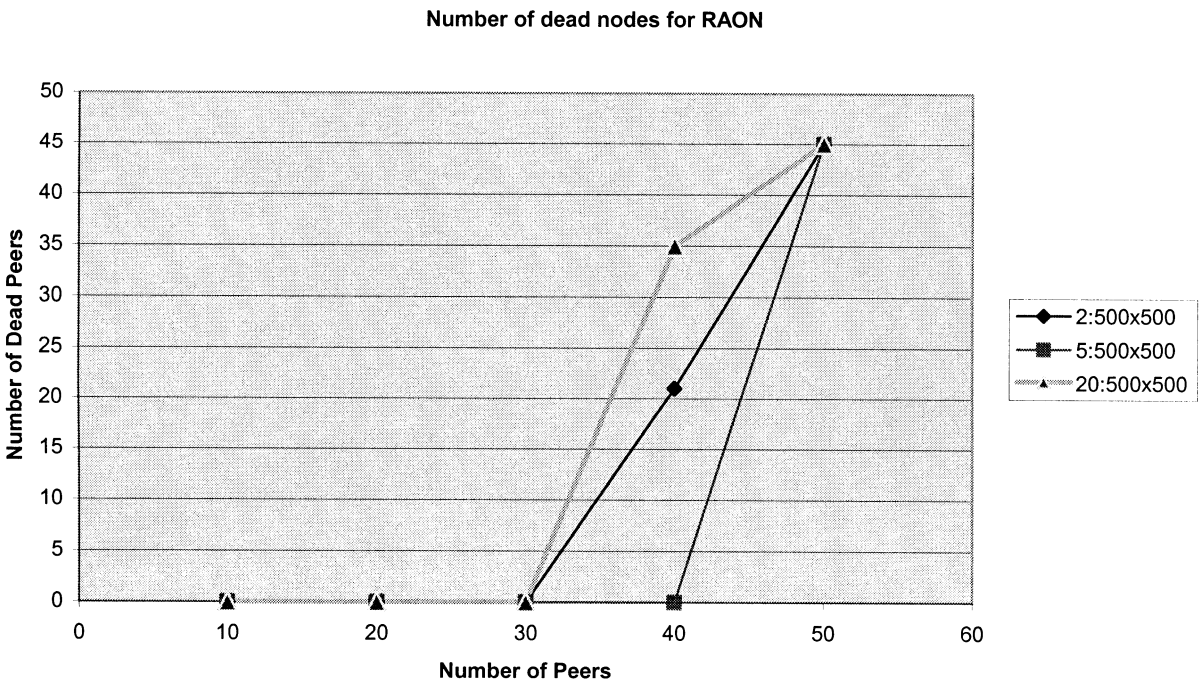

Figure 4.21 Number of dead nodes for RAON

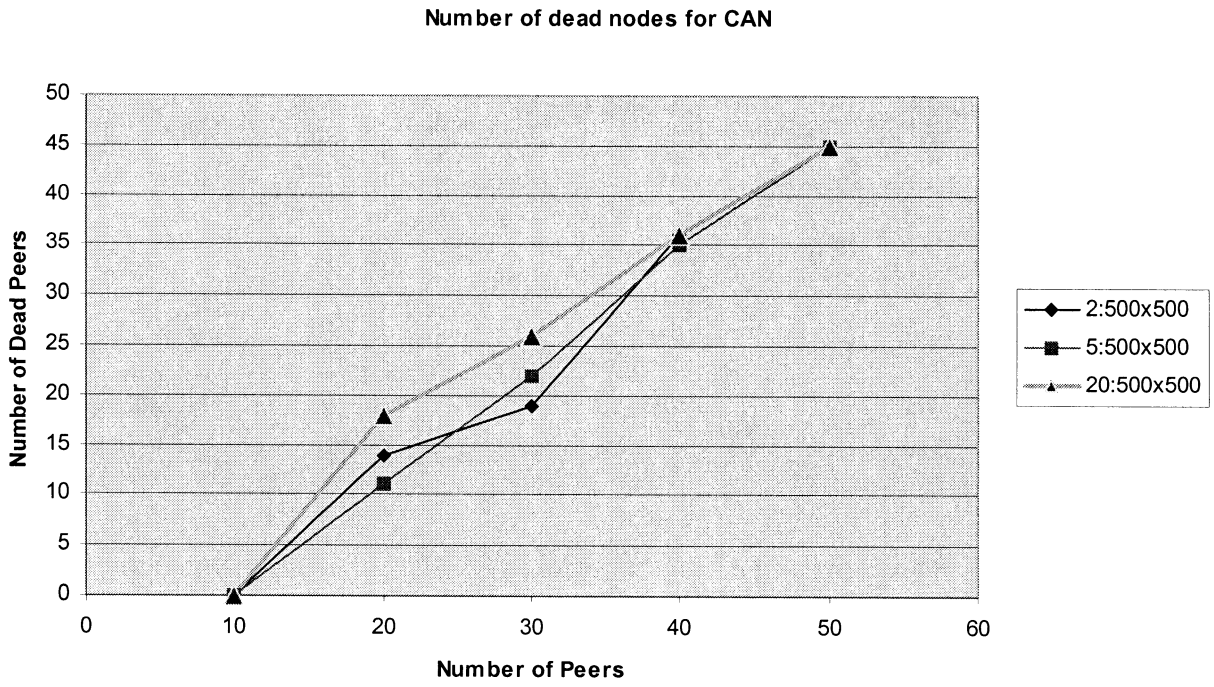

Figure 4.22 Number of dead nodes for CAN

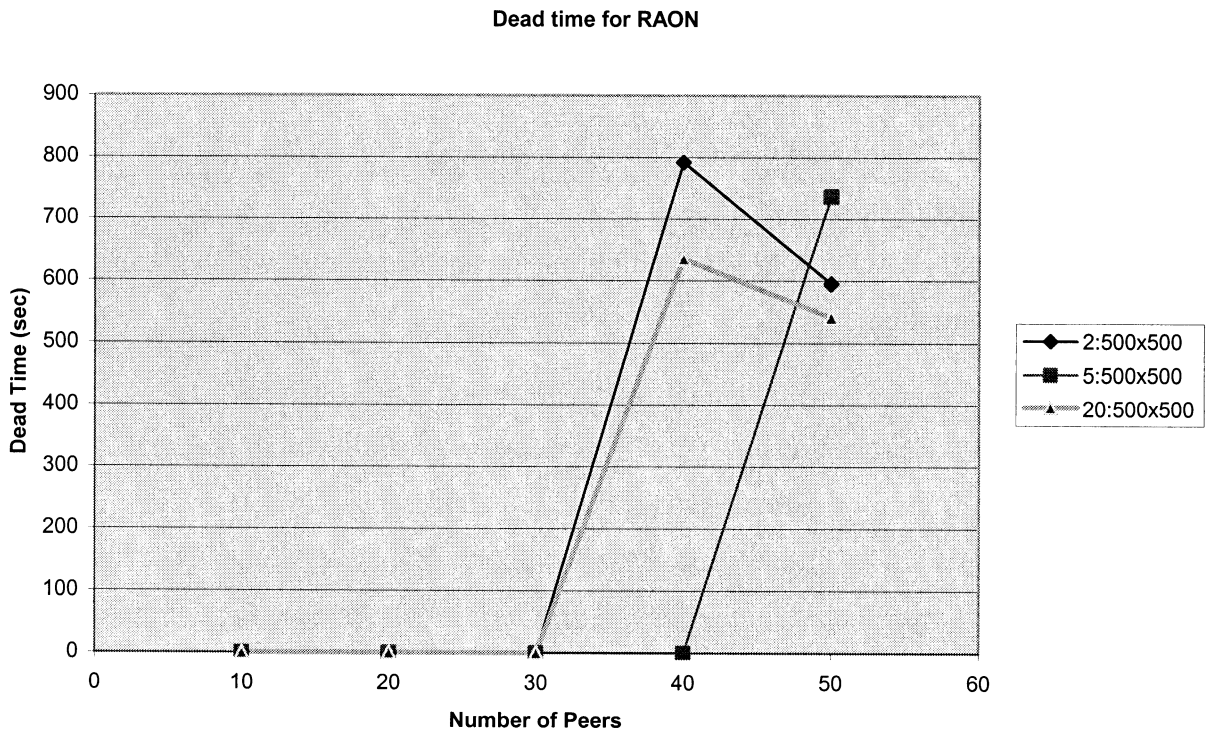

Figure 4.23 Dead time for RAON

In case of CAN, Figure 4.24, the graph shows a rapid increasing in number of dead nodes by increasing the number of peers.

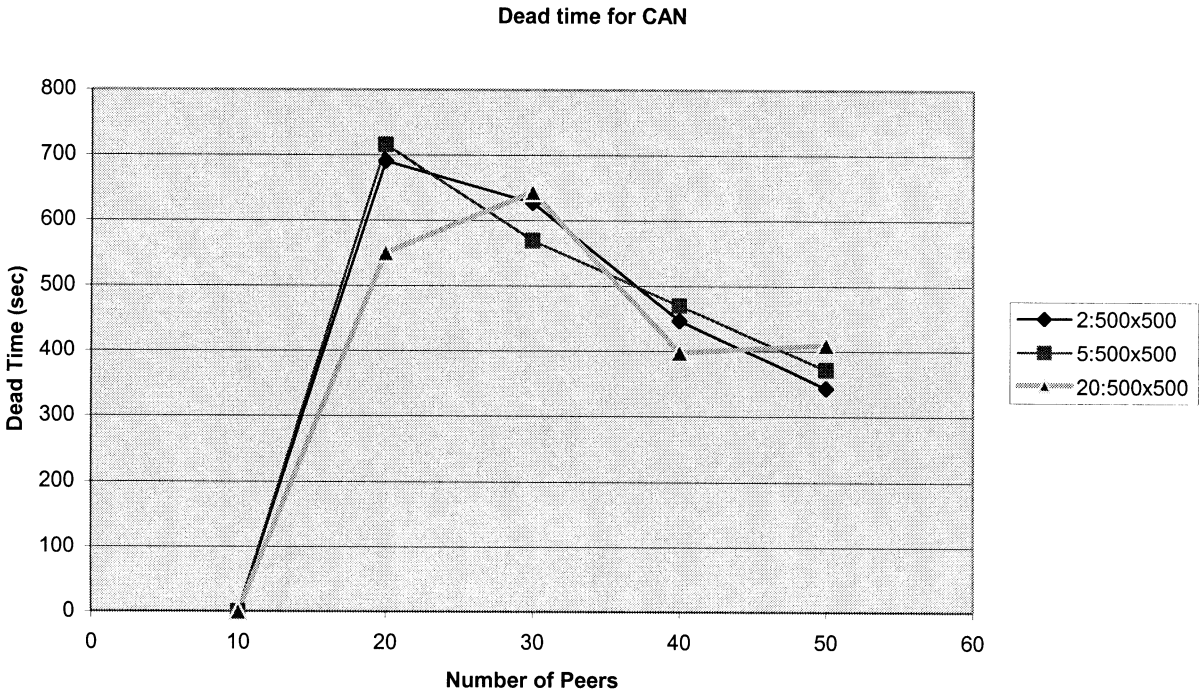

Figure 4.24 Dead time for CAN

It is noticed in Figure 4.24 that CAN runs out of power very fast, effect that is shown in increasingly low values of dead time.

Compared with RAON, CAN is running out of power rapidly, which means a higher level of energy consumption.

Chapter 5 Conclusions

The overlay network approach shows promising results in its ability to deploy various applications, such as P2P, multicast and VoIP. The goal of this project was to show the differences and characteristics of two approaches in P2P overlay networks on ad-hoc mobile networks – structured and unstructured. The two file sharing P2P systems were chosen to analyze. One is RAON (Resource-Aware Overlay Network), which is an improved variant of Gia and the other is CAN (Content-Addressable Network).

RAON design is based on Gia. Gia is an unstructured P2P system with superior performance over other unstructured systems due to its flow control and biased random walk forwarding schemes. The design of Gia was improved to address problems specific to MANET. RAON comes with important features in terms of making forwarding decisions based on the MANET characteristics of link instability and power constraints. Two new features were added to Gia's basic biased random walk forwarding: NCS (Neighbor Coloring Scheme) and PNR (Proactive Neighbor Replacement). The NCS uses a ranking system that ranks neighbors according to their available resources. It monitors the condition of the overlay links by sending update messages and measuring the delay periodically. It colors the links either green, yellow or red depending upon the link latency and the neighbor's residual power. The PNR algorithm attempts to find a better node to replace a neighbor if the connection to the neighbor is found to be unstable.

CAN design is based on the work presented in [11], which is though not designed for MANET network the algorithm presents sufficient features for adaptability to ad-hoc mobile networks. CAN is a structured system based on DHT (Distributed Hash Tables). DHT design goal is to achieve a very high level of scalability. In the same time, the rapid arrival and departure of nodes has to be handled without system disruption. CAN is a good example in the category of structured system with its completely distributed architecture, high scalability and with increased level of fault-tolerance. The design for CAN is based on a 2-dimensional Cartesian coordinate space. Some of the design improvements proposed in CAN such as multi-dimensioned coordinate spaces, multiple realities or multiple coordinate spaces and multiple hash functions presented in [11] were not implemented in our comparison simulations. The CAN system was kept at the basic level to show the normal behavior in a mobile environment. The

implementation in our simulation uses a uniform distribution of nodes based on the largest size area to be split. Thus, the zone that is split to accommodate the new node is the one with the largest size. A uniform partitioning of the space is highly desirable to achieve load balancing.

The performance of RAON and CAN was evaluated using a packet-level simulator called NS2. Simulation results show that RAON is able to achieve more stable and improved query success rate and lower query delay as compared to CAN under a variety of topology and load conditions. The results show that RAON has a greater advantage in terms of performance because of its implementation features designed for mobile environments.

Reference List

[1] Balakrishnan, Kaashoek, Karger, Morris, Stoica. Looking up data in P2P systems. *In COMMUNICATIONS OF THE ACM February 2003/Vol. 46, No. 2. February 2003.*

[2] J. P. Macker and M. S. Corson, "Mobile Ad Hoc Networking and the IETF", *IEFT Mobile Ad Hoc Networks (MANET) Working Group Charter*, http://www.ietf.org/html.charters/manet-charter.html

[3] E. Royer and C-K Toh, "A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks", in *IEEE Personal Communications Magazine*, pages 46-55, April 1999.

[4] C. E. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers", in *Proceedings of the ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pages 234-244, August 1994.

[5]. J. Guterman. Gnutella to the Rescue? Not so Fast, Napster fiends. Link to article at http://gnutella.wego.com, Sept. 2000.

[6] Maymounkov, Mazières. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. *In ACM Computing Surveys archive Volume 36, Issue 4. December2004.*

[7] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks" in *Mobile Computing*, pages 153-181. Kluwer Academic Publishers, 1996.

[8] C. E. Perkins and E. M. Royer "Ad-hoc On-Demand Distance Vector Routing", in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99)*, pages 90-100, February 1999.

[9] "Napster". http://www.napster.com.

[10] "Gnutella". http://www.gnutella.com.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network", in *Proceedings of SIGCOMM 2001*, August 2001.

[12] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications", in *Proceedings of SIGCOMM 2001*, August 2001.

[13] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems", in *Proceedings of ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329-350, November 2001.

[14] B. Y. Zhao, J. Kubiatowicz, and A.D. Joseph, "Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing", *Technical Report UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley*, 94720, April 2001.

[15] M. A. Jovanovich, "Modelling Large-Scale Peer-to-Peer Networks and a Case of Study of Gnutella", *Master's thesis, Department of Electrical and ComputerEngineering and Computer Science, University of Cincinnati*, June 2000.

[16] S. Androutsellis-Theotokis, "A Survey of Peer-to-Peer File Sharing Technologies", *White Paper: Athens University of Economics and Business, Greece*, 2002

[17] "KaZaA". http://www.kazaa.com.

[18] I.Clarke, O.Sandberg, and B.Wiley. Freenet: A distributed anonymous information storage and retrieval system. In Proceedings of the Workshop on Design Issues in Anonymity and Unobservability, Berkeley, California, June 2000.

[19] S. Sen, and J. Wang, "Analyzing Peer-to-Peer Traffic Across Large Networks", in *Proceedings of the ACM SIGCOMM Internet Measurement Workshop 2002*, November 2002.

[20] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-like P2P Systems Scalable", in *Proceedings of SIGCOMM 2003*, August 2003.

[21] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-level Multicast using Content-Addressable Networks", in *Proceedings of NGC*, 2001.

[22] ADAMIC, L. A., LUKOSE, R. M., PUNIYANI, A. R., AND HUBERMAN, B. A. Search in Power-law Networks. *Physical Review E 64* (2001).

[23] M. Castro, M. B. Jones, A-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang and A. Wolman, "An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer overlays", in *IEEE Infocom 2003*, April, 2003.

[24] SAROIU, S., GUMMADI, P. K., AND GRIBBLE, S. D. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN'02)* (San Jose, CA, Jan. 2002).

[25] Tang, Xu, Mahalingam. pSearch: Information Retrieval in Structured Overlays *In ACM SIGCOMM ComputerCommunication Review, Volume 33, Issue 1 (Jan 2003)*

[26] R. Kravets and P. Krishnan, "Application-driven power management for mobile communication", in *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 263-277, 2000.
[27]S. P. Ratnasamy, "A Scalable Content-Addressable Network", *PhD's thesis, University of California at Berkeley* ,Fall 2002.
[28] P. Reynolds, and A. Vahdat, "Efficient Peer-to-Peer Keyword Searching", *Technical report, Duke University, Durham, NC*, 2002
[29] Q. Lv, S. Ratnasamy, and S. Shenker, "Can Heterogeneity Make Gnutella Scalable?", in *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02).* MIT Faculty Club, Cambridge, MA, USA, March 2002.
[30] Q. Lv, P. Cao, E. Cohne, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks", in *Proceedings of 16th ACM International Conference on Supercomputing (ICS'02)*, November 2001.
[31] P. Francis. Yoid: Extending the Internet Multicast Architecture. Unpublished paper, available at http://www.aciri.org/yoid/docs/index.html, Apr. 2000.
[32] D. Tsoumakos and N. Roussopoulos, "Analysis and Comparison of P2P Search Methods", *University of Maryland, Dept. of Computer Science Technical Report (CS-TR-4539),* November 3, 2003.
[33] Z. Ge, D. R. Fiegueiredo, S. Jaiswal, J. Kurose, and D. Towsley, "Modeling peer-to-peer file sharing systems", in *IEEE Infocom 2003*, April 2003.
[34] Q. He, M. Ammar, G. Riley, H. Raj, R. Fujimoto, "Mapping Peer Behavior to Packet-level Details: A Framework for Packet-level Simulation of Peer-to-Peer Systems", in *MASCOTS 2003*, October 2003.
[35] "NS2". http://www.isi.edu/nsnam/ns/.
[36] Sung-Ju Lee, Julian Hsu, Russell Hayashida, Mario Gerla, and Rajive Bagrodia, "Selecting a Routing Strategy for Your Ad Hoc Network", *Computer Communications,* special issue on Advances in Computer Communications and Networks: Algorithms and Applications, Vol. 26, Issue 7, pages723-733, May 2003.
[37] G. Lau, "Resource-Aware Topology Adaptation in P2P Overlay ADHOC Network", *Master's thesis, Department of Electrical and Computer Engineering,, Ryerson University*, 2004.
[38] Rhea, Geels, Roscoe, Kubiatowicz. Handling Churn in a DHT. *University of California, Berkeley and Intel Research, Berkeley. December 2003.*
[39] Ramasubramanian, Sirer. Proactive Caching for Better than Single-Hop Lookup Performance. Cornell University,Computer Science Department Technical Report TR-2004-1931. February 2004.
[40] S. Corson and J. Macker, "Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations," RFC 2501, January 1999, Category: Informational, work in progress. [Online]. Available: http://www.ietf.org/rfc/rfc2501.txt.
[41] Z. J. Haas and M. R. Pearlman. The zone routing protocol for ad hoc networks, 2000. Internet Draft.
[42] Cohen, E. and Shenker, S. Optimal replication in random search networks. Preprint 2001.
[43] GOYAL, P., VIN, H. M., AND CHENG, H. Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks. In *Proceedings of ACM SIGCOMM '96* (Stanford, CA, Aug. 1996).