1-1-2012

Resource-Aware Cooperative Caching on Mobile Ad-hoc Peer to Peer Networks

Hoda R.K. Nejad
Ryerson University

Resource-Aware Cooperative Caching on Mobile Ad-hoc Peer to Peer Networks

by

Hoda Razavi Khosravani Nejad

A master thesis presented to
Ryerson University
in partial fulfillment of the requirements
of the degree of

Master of Applied Science

Department of Computer Network
Ryerson University
Toronto, Ontario, Canada
May 2012

Thesis advisor

Dr. Muhammad Jaseemuddin

Author

Hoda Razavi Khosravani Nejad

Resource-Aware Cooperative Caching on Mobile Ad-hoc Peer to Peer Networks

Abstract

With the emergence of wireless devices, service delivery for ad-hoc networks has started to attract a lot of attention recently. Ad-hoc networks provide an attractive solution for networking in the situations where network infrastructure or service subscription is not available. We believe that overlay networks, particularly peer-to-peer (P2P) systems, is a good abstraction for application design and deployment over ad-hoc networks. The principal benefit of this approach is that application states are only maintained by the nodes involved in the application execution and all other nodes only perform networking related functions. On the other hand, data access applications in Ad-hoc networks suffer from restricted resources. In this thesis, we explore how to use Cooperative Caching to improve data access efficiency in Ad-hoc network. We propose a **R**esource-**A**ware **C**ooperative **C**aching P2P system (RACC) for data access applications in Ad-hoc networks. The objective is to improve data availability by considering energy of each node, demand and supply of network. We evaluated and compared the performance of RACC with Simple Cache, CachePath and CacheData schemes. Our simulation results show that RACC improves the delay of query as well as energy usage of the network as compared to Simple Cache, CachePath and CacheData.

Acknowledgments

It would not have been possible to write this Master thesis without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here.

This thesis would not have been possible without the help, support and patience of my supervisor Professor Muhammad Jassemudin. The good advice, support and friendship of Dr. Boby Ma, has been invaluable on both an academic and a personal level, for which I am extremely grateful.

I would like to thank my husband Arash for his personal support and great patience at all times. My parents and sister have given me their unequivocal pray and support throughout, as always, for which my mere expression of thanks likewise does not suffice.

Last, but by no means least, I thank my friends at Ryerson university and Sharif university, specially Saeed Rashwand, for their support and encouragement throughout.

Contents

List of Figures

List of Tables

Table : List of Abbreviations

Abbreviation	**Full Word**
ACK	Acknowledgment
AODV	Ad hoc On-Demand Distance Vector
CARP	Cache Array Routing Protocol
CDF	Cumulative Distributed Function
CPU	central processing unit
DHT	Distributed Hash Table
DRT	Distributed Routing Table
DSDV	Destination-Sequenced Distance-Vector Routing
DSR	Dynamic Source Routing
GLDU	Greedy Dual Least Utility
GUID	Globally Unique Identifier
ICP	Internet Cache Protocol
LAN	Local Access Network
MANET	Mobile Ad-hoc Network
MP3	MPEG Audio Layer III
NCS	Neighbor Coloring Scheme
NED	NEtwork Description
OMNET++	Objective Modular Network Test-bed in C++
PDA	Personal Digital Assistant
P2P	Peer-to-Peer
PSM	Power Saving Mode
RACC	Resource Aware Cooperative Cache
RAON	Resource Aware Overlay Network
RE	Residual Energy
RTT	Round Trip Time
SFQ	Start-time Fair Query
TTL	Time to Live

Chapter 1

Introduction

The Internet is an integrated cooperative network of millions of nodes around the world, which share the resources. Today there are a large number of applications which tend to use the network, consume bandwidth, and send packets far. Internet was originally formed as a Peer-to-Peer (P2P) system in the late 1960s. However, it has become mostly a client/server-base network with millions of nodes. The current peer-to-peer applications use Internet as it was originally intended: as a medium of communication for computers to share resources with others. In 2000, through Napster, a music sharing application, millions of users connected to Internet and started using their home computers for more than just web-surfing. P2P networks allow direct sharing of resources (CPU, bandwidth, storage) with a large number of users in a decentralized manner. These networks can serve many purposes from distribution of digital content (such as Napster, Gnutella and BitTorrent) to Internet telephony (for example Skype). Unlike client-server model, where a limited number of servers provide content to many clients, P2P networks eliminate the need for central servers. These networks focus on direct communication between peers acting as both clients and servers

and share resources as equals. Due to the transient population of P2P networks, their topology is dynamic, in which peers often join and leave the network. Most current P2P architectures are primarily designed for the traditional wired infrastructure.

The rapid growth of wireless communications and mobile computing technologies has led to the emergence of a new network concept, known as Mobile Ad-hoc NETworks (MANETs). These networks contain a set of mobile nodes that do not depend on existing fixed infrastructure for operation. The rapid deployment characteristics and self-organization of ad-hoc networks are appropriate for the battlefield and disaster recovery. Due to the lack of infrastructure, each node in the network acts as a router, transmitting data packets for other nodes. The data access is very important since the goal of using ad-hoc networks is to provide access to information of mobile nodes. MANETs, in contrary to traditional communication networks, are fully decentralized and have dynamic topologies. However, finite energy sources of mobile nodes limit the network lifetime. Regardless of the communication protocol, researchers must choose the communication mode: single-hop or multi-hop. In a single-hop communication, each node sends its data directly to the base station. In a multi-hop connection, each node sends data destined ultimately to the base station through intermediate nodes. An ad-hoc network using the multi-hop communication where there is no central access point.

Operating in ad-hoc mode allows all mobile devices within range of each other to discover and communicate in a peer-to-peer model without using central access points.

The similarities between ad-hoc and P2P networks provide an interesting research topic. Both networks follow a P2P model characterized by the absence of a central node or a control server; all participants work together for the whole system to work. A key issue in both

systems is the process of finding the requested data or a route effectively in a decentralized manner. Ad-hoc and P2P networks also have dynamic topologies: nodes constantly move in and out of radio coverages of neighbouring nodes and their peers continuously join and leave the network.

Although ad-hoc and P2P networks are similar, the integration of these two concepts presents significant challenges. P2P networks aim to provide resource sharing on top of an existing communication infrastructure, such as Internet, where forming an overlay network and maintenance of connections among the peers are important. Ad-hoc networks focus on providing multi-hop mobile connectivity where no infrastructure exists or the infrastructure is insufficient. P2P applications run at the application layer of the OSI model, while ad-hoc networks operate at the network layer or data link layer. Routing functionality is present in both layers. On one hand, the application layer routing is used by applications to find a peer which contains the desired information within the P2P overlay network. On the other hand, the network layer or data link layer routing is used to discover the route to a given peer. Another difference is that the nodes in ad-hoc networks are usually mobile nodes which are constrained in bandwidth and energy, which is usually not a problem for wired P2P network connect to Internet applications.

These differences result in significant challenges regarding to the integration of these two concepts - the existing P2P systems form an overlay network that is totally independent of the underlying physical topology. Overlay network differs from underlay network as neighbours in the P2P overlay network may actually be located on opposite sides of the world. This can lead to extremely long routes and unnecessary traffic.

1.1 Motivation

For the places where the access to Internet is hard decentralized communication and the use of MANETs becomes important. They can be used to allow villagers to communicate or aid workers to share documents. Given the similarities between P2P systems and MANETs, it is clear that the P2P networks are particularly adapted to allow the distribution of content and communication in this environment.

Although routing is an important issue in ad-hoc networks, but the ultimate goal of these networks is to provide mobile nodes with access to information. However, MANETs are limited by discontinuous network connections, limited power and resources. These restrictions raise several new challenges for data access applications with respect to data availability and access efficiency. In mobile ad hoc networks, due to frequent network partition, data availability is lower compared to traditional wired networks. **cooperative caching** offers a solution to this problem. Cooperative caching is a technique which allows sharing the resources among mobile nodes. However, limited cache space, movement of nodes and frequent disconnections limit availability of data. By caching frequently accessed data in the MANETs, we can improve data access performance and availability[1]. Due to mobility and resource constraints of MANET, the caching techniques which were designed for wired networks may not be applicable to these networks.

Once MANET is formed, the next challenge is delivery and discovery of ad-hoc services for users and devices to communicate over these networks. It faces many challenges which are posed by instability of the links, node transience, unexpected disconnection and limited resources.

I consider the overlay network at the application level. A collection of networks to

create a virtual topology above the physical topology that is independent of underlying network and facilitates the data searching (Fig. 1.1). Each connection between nodes is designated as an overlay virtual link and each link may consist of multiple physical hops. The proposed scheme focuses more on the data discovery rather than data delivery. In this approach overlay nodes are aware of the overlay topology. Any changes in the topology of the underlying network are transparent to them. In fact, the overlay network is an alternative key for the introduction of new services that were too difficult to deploy at the IP level.

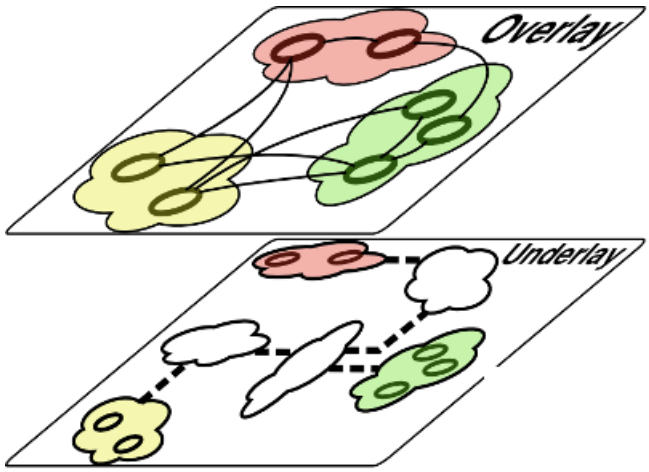

Figure 1.1: An example of overlay on top of ad-hoc network. Overlay nodes do not need to be aware of the intermediate nodes in the Ad-hoc topology.

Due to the similarities in the dynamic topology and lack of infrastructure between the MANET and P2P networks, it is interesting to design an architecture which adapts P2P for MANET environment. However, most of existing P2P systems are designed for wired networks, thus limited power has not been taken into account. The main challenges would be to efficiently locate the desired content in the MANET.

1.2 Objective

The purpose of this thesis research is to design a Resource-Aware Cooperative Cache(RACC) scheme that fits well for the mobile ad-hoc network. Due to the constrained resources of mobile nodes, hot spots must be avoided in order to prevent congestion or node failure. Thus, the P2P overlay architecture is chosen to serve this purpose. On the other hand, one of the key challenges in ad-hoc peer to peer networks is maximizing the network's lifetime. It will significantly reduce communication energy consumption and extend the network's lifetime to choose appropriate communication mode. However, existing P2P systems are designed for wired nodes, thus problems such as link instability and constrained power were not addressed. RACC is proposed to improve content availability and content discovery efficiency through exposing node resource constraints to application level overlay network. Furthermore, In order to decrease network energy consumption and delay, it is desirable that the proposed caching mechanism takes in to account the demands and supply of the network and energy of each intermediate node while making caching decision.

The objective of this mechanism is to reduce the energy consumption of the whole network and reduce the delay which a node faces for transferring a specific data packet.

The following are the objectives of the thesis:

1. Research on existing P2P systems and their adaptability to ad-hoc network.

2. Designing an overlay P2P architecture where peers are aware of Residual Energy of their neighbors.

3. Designing a cooperative cache policy which considers energy of nodes, demand and supply of the network to distribute data in the system and reduce the delay and energy

usage of network.

4. Performing a simulation for the design.

1.3 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 provides some background information on Cooperative Caching scheme in wire and wireless networks, P2P networks and their applications, and energy conservation algorithms. In Chapter 3, the design issues of overlay P2P for ad-hoc networks are addressed. Gia, an existing P2P system, as well as proposed Resource-Aware Cooperative Caching scheme are described in this chapter. We then demonstrate simulation results and evaluate the performance of different cache policies over MANET. This chapter contains analysis of RACC over mobile ad-hoc network. Finally, Chapter 5 concludes the thesis with a summary of the research findings and some recommendations for future works.

Chapter 2

Background

This chapter provides the fundamental knowledge of the technologies used in this thesis. Section 2.1 gives an overview of Cooperative Caching and ad-hoc Caching policies. Section 2.2 is an overview of RAON algorithm. On Section 2.3 we describe some of the existing P2P architectures used for file sharing applications. Finally, Section 2.4 discusses power conservation techniques.

2.1 Cooperative Caching

Recent years have seen a rapid development of mobile computing and wireless communication techniques. However, constraints such as limited bandwidth, frequent network disconnections, and limited local resources (energy) remain to be overcome for effective data access in wireless network environments. Data caching in mobile nodes has been shown to be effective for improving the system performance, i.e. the terminal latency and

energy consumption. However, the caching on individual systems cannot be scaled easily. Cooperative caching based on the idea of sharing and coordination of cached data among multiple users can be particularly effective for information access in ad-hoc networks.

Cooperative caching allows peers to fetch data from each other instead of from the server. In case a peer receives a request for an object which is not stored locally, it tries to locate the requested object in it's peers caches. If this search wasn't successful, the request is redirected to the server which has the requested object. The main goals of the cooperative caching are to reduce the load on the server and to improve peer-perceived latencies.

Peer-to-peer cooperative caching suggests content providers with the assure of delivery of data to a large population of users without the need for significant investment in servers. However, the design of efficient algorithms for distributed caching is important when network nodes have limited memory and energy. In addition, due to mobility and resources constraints in ad-hoc networks, cooperative caching techniques designed for wired networks may not be appropriate for MANETs. In this thesis, we consider cooperative caching mechanism in order to minimizing the total energy usage and delay in MANETs. The following subsections discuss some of the existing caching policies for wired and wireless ad-hoc networks.

2.1.1 Cooperative Caching in Wired Network

In Cooperative Caching method multiple nodes share and coordinate cached data. This scheme is widely used to improve the Web performance in wired networks. In the following we provide additional information about recent research focusing on cooperative

caching approaches for wired networks. Existing cooperative caching schemes for the Web environment can be classified as message-based, directory-based, and hash-based.

Duane Wessels and Kim Claffy[2] introduced the Internet Cache Protocol (ICP). As a message-based protocol, ICP supports communication between caching proxies using a simple query-response dialogue. Directory-based protocols for cooperative caching, such as Cache Digests and Summary Cache, let caching proxies exchange information about cached content. The Cache Array Routing Protocol (CARP)[3] is the most notable hash-based cooperative caching protocol. The rationale behind CARP constitutes load distribution by hash routing among Web proxy cache arrays.

Because these protocols often assume fixed network topology and cause high computation and communication overhead, they are unsuitable for ad-hoc networks.

2.1.2 Cooperative Caching in Wireless Ad-hoc Networks

Demand for instant sharing of resources, such as music files and photos captured by user terminals, have attracted a lot of attention in ad-hoc networks. P2P file-sharing has become popular in wired networks. However, mobile networks have some limitations in terms of access bandwidth, users' mobility and user terminals' capacity.

To tolerate network partitions and improve data accessibility, Takahiro Hara [4] proposed several replica allocation methods for ad-hoc networks. In Takahiro 's schemes, a node maintains replicas of frequently requested data. The data replicas are periodically relocated based on two criteria: the access frequency from each mobile host to each data item and overall network topology. Although data replication can improve data accessibility, sig-

nificant overhead is associated with maintaining and redistributing the replicas, especially in ad-hoc networks. Maria Papadopouli and Henning Schulzrinne [5] proposed7DS architecture similar to cooperative caching. This architecture defines two protocols to share and disseminate data among users. It operates on a pre-fetch mode to gather data for serving the users' future needs or on an on-demand mode to search for data on a single-hop multicast basis. The 7DS architecture focuses on data dissemination instead of cache management. Further, it focuses on a single-hop rather than a multi-hop environment. However, resource constraints and node mobility limit the applications of these techniques in ad-hoc networks. In order to utilize the server and the peer resources for efficient data transmission, Kozat and ztan Harmanc [6] proposed a cooperative hybrid P2P video on-demand architecture. This scheme utilizes the server and the peer resources for efficient transmission of popular videos by considering future demand and supply of network. They propose the cooperative caching problem as a supply-demand based utility optimization problem. In their system architecture, each peer dedicates some cache space to store a particular segment of a video file as well as some of its upload bandwidth to serve the cached segment to other peers. Peers join the system and issue a streaming request to a control server. The control server directs the peers to streaming servers or to other peers who have the desired video segments. The control server also decides about the video segments which each peer should caches. Liangzhong Yin and Guohong Cao [7] proposed caching techniques; CachePath, CacheData, and HybridCache. These techniques deploy the underlying routing protocols to overcome the limitation of ad-hoc networks. They improve the performance by caching the data locally or caching the path to the data to save storage. To increase data accessibility mobile nodes should cache different data items compared to their neighbours. Although

this increases data accessibility, it also can increase query delays because the nodes might have to access some data from their neighbours instead of accessing it locally. Bin Tang and Himanshu Gupta [8] design an efficient caching technique for ad-hoc networks with memory limitations. Essentially, they develop efficient strategies to select data items to cache at each node. In their network model, there are multiple data items; each data item has a server, and a set of clients that wish to access the data item at a given frequency. Each node carefully chooses data items to cache in its limited memory to minimize the overall access cost. Jing Zhao and Ping Zhang [9] presented their design and implementation of Layered cooperative cache in wireless P2P networks, where the cooperative cache layer is on top of the network layer (TCP/IP). The replacement algorithm and client query model is similar to their previous studies in [7]. They improve their previous study by proposing a novel asymmetric cooperative cache approach. In the developed approach the data requests are transmitted to the cache layer on every node, but the data replies are only transmitted to the cache layer at the intermediate nodes that need to cache the data. To reduce the end-to-end delay they allows data pipelines.

As our best knowledge none of the previous studies consider energy consumption, demand and supply at the same time. In this thesis we develop the study in [7] in a way which intermediate nodes consider these 2 important factors for making caching decision in the network.

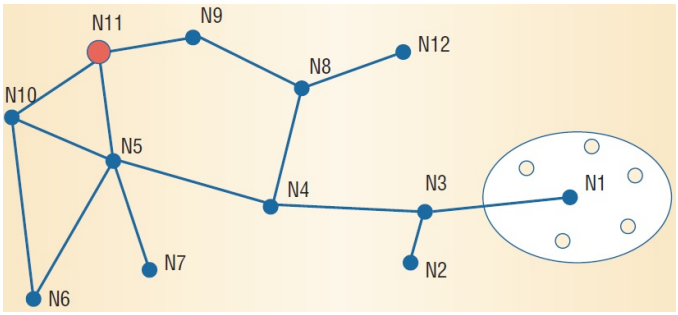

Figure 2.1: System model of Ad-hoc network

2.1.2.1 System Model

Fig. 2.1 is part of an ad-hoc network which Liangzhong Yin and Guohong Cao proposed in [7]. Some nodes in the ad-hoc network may have wireless interfaces to connect to the wireless infrastructure such as wireless LAN or cellular networks. Suppose node N_{11} is a data source (center), which contains a database of n items d_1, d_2, ..., d_n. In ad-hoc networks, a data request is forwarded hop-by-hop until it reaches the data center and then the data center sends the requested data back. To reduce the bandwidth usage and the query delay, the number of hops between the data center and the requester should be as small as possible. In the following, three basic cooperative caching schemes of CacheData, CachePath and HybridCache are introduced.

2.1.2.2 CachePath Concepts

To understand concept of CachePath the following example is used. Suppose node N_1 requests a data item d_i from N_{11}. When N_3 forwards d_i to N_1, N_3 knows that N_1 has a copy of the data. Later, if N_2 requests d_i, N_3 knows that the data source N_{11} is three hops away

whereas N_1 is only one hop away. Thus, N_3 forwards the request to N_1 instead of N_4. Many routing algorithms provide the hop count information between the source and destination. Caching the data path for each data item reduces bandwidth and power because nodes can obtain the data using fewer hops. In CachePath, a node does not need to record the path information of all passing data. Rather, it only records the data path when it's closer to the caching node than the data source. For example, when N_{11} forwards d_i to the destination node N_1 along the path N_5 N_4 N_3, N_4 and N_5 do not cache d_i's path information because they are closer to the data source than the caching node N_1.

2.1.2.3 CacheData Concepts

In CacheData, the node caches the data instead of the path when it finds that the data is frequently accessed. For example, in Fig. 2.1, if both N_6 and N_7 request d_i, N_5 might think that d_i is popular and cache it locally. N_5 can then serve N_4's future requests directly. Because the CacheData approach needs extra space to save the data, it should be used prudently. Suppose N_3 forwards several requests for d_i to N_{11}. The nodes along the path N_3, N_4 and N_5 might want to cache d_i as a frequently accessed item. However, they waste a large amount of cache space if they all cache d_i. To avoid this, CacheData enforces another rule: A node does not cache the data if all requests for the data are from the same node. In this example, all the requests N_5 received were from N_4, and those requests in turn came from N_3. With the new rule, N_4 and N_5 won't cache d_i. If N_3 receives requests from different nodes, for example, N_1 and N_2, it caches the data. If the requests all come from N_1, N_3 doesnt not cache the data but N_1 does. Certainly, if N_5 later receives requests for d_i from

N_6 and N_7, it can also cache the data.

2.1.2.4 HybridCache

To further improve the performance of their cooperative cache network, Liangzhong Yin and Guohong, proposed HybridCache scheme. Hybrid scheme exploits the strengths of CacheData and CachePath and avoids their weaknesses. Specifically, when a mobile node forwards a data item, it caches the data or path based on some criteria. These criteria include the data item size S_i and the TTL time TTL_i. For a data item d_i, they use the following heuristics to decide whether to cache data or the path:

- If S_i is small, CacheData is optimal because the data item only needs a small part of the available cache; otherwise, CachePath is preferable because it saves cache space.
- If TTL_i is small, CacheData is preferable. Because the data item might soon be invalid, using CachePath can result in choosing the wrong path and having to resend the query to the data center. For large TTL_is, however, CachePath is acceptable.

2.2 RAON Design

Resource-Aware Overlay Network or RAON uses Gia as the foundation for its design [10]. RAON adds the features required to address issues of adaptability of Gia in MANET. RAON approach modifies Gia's design to which performs query forwarding decision based on link instability and power constraints in MANET. RAON incorporates all the design features of Gia but modifies the biased random walk to incorporate dynamic factors in for-

warding a query such as link instability and node power constraints. It defines the capacity of a node to be a static function of CPU speed and memory. It removes the bandwidth of a node from the definition of capacity because the bandwidth is no more a static factor in MANET. RAON introduces a ranking system to classify its neighbors according to their dynamic properties. The proposed solution is Neighbor Coloring Scheme (NCS) is used by each node to keep track of the neighbors' energy levels and the delay it experiences with each of its neighbors. The NCS colors a link based on round trip time (RTT) and the neighbors' energy.

In RAON a node measures the stability of a link to its neighbor by measuring the RTT on the link. It also receives the residual power level (energy) of its neighbors. A node can also determine energy level by periodically exchanging energy update messages with its neighbors. They define three energy states of HIGH, MEDIUM, and LOW. In the implementation of NCS each node monitors its energy level and determines its energy state relative to its battery size, and updates its neighbors only when state transition occurs.

The NCS colors a link according to RTT and the neighbor's energy into GREEN, YELLOW, or RED(Table. 2.1). The modified biased random walk algorithm for a node in RAON considers the neighbor color in addition to the neighbor's capacity before forwarding a query to the neighbor.

Table 2.1: RAON Neighbour Color Function.

Neighbor Color	condition
GREEN	$Energy = HIGH \&\& RTT = LOW$
YELLOW	$(Energy = (Medium \&\& RTT != HIGH) \mid\mid (RTT = MEDIUM \&\& Energy != LOW)$
RED	$Energy = LOW \mid\mid RTT = HIGH$

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 before. Among the ”available” neighbors, it groups them according to their color and selects the one with the greatest capacity from the high color group. 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 RTT. The probe message can be a separate control message or piggybacked on other messages. After a node sent out a probe message to its neighbors, it sleeps for a probe-interval seconds and waits for an ACK message in the meantime. 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 only when there is a transition. 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 and link delay.

2.3 Peer-to-Peer Overlay Networks

In a pure peer to peer system, computers communicate directly with each other and share information and resources without using dedicated servers. In this overlay network each peer only maintains information of its neighboring nodes in the overlay. The peers share their resources and content and forward data from other peers. Hosts may join or leave the P2P network at any time they want, resulting in rapidly changing network topologies. A common characteristic of these new systems is that they build up a virtual network on top of the physical network. P2P is in application layer. This virtual network has its own routing mechanisms which has a significant impact on application properties such as performance, reliability. Among all applications, file sharing is one of the most used applications in such a network. In this section, we describe some of the architecture designed for this purpose.

2.3.1 Introduction to P2P File-Sharing Systems

In recent years, P2P systems for file sharing attracted attention of a lot of researchers. By definition, P2P is a collection of computers that cooperate to form an overlay network. Napster [11]is the first system of P2P file sharing. Although it is a system of file sharing, it's not completely a P2P network because it uses a central server to store all the index files. The index file is a file pointer that indicates where files are located.

Even Napster is not a true P2P system; it started the revolution of P2P file sharing, which attracts the attention of many researchers to start developing more advanced P2P architectures. Typically, P2P file sharing allows users to share files with other users who

are referred as peers. Peers join to form a network and find the desired files and download them directly from other peers.

In a P2P network each peer is a server and a client, and there is no single point of failure. The node which has downloaded the content becomes a content server itself; therefore the servers are distributed across the network. This is in contrast with the traditional client-server model where one server serves all customers. P2P systems are usually categorized by how the P2P network is constructed and how to find the desired content within it. The transmission of files usually happens via a direct connection between the source and destination.

The existing P2P systems can be classified into two categories: structured and unstructured systems. Unstructured P2P systems like Gnutella [12] do not have a particular way to find the desired content; they typically flood the network to search the entire network. On the other hand, structured P2P systems often use distributed hash tables directly to find peers that contain the desired content. Thus, structured P2P systems can preserve a lot of bandwidth compared to unstructured P2P systems. Some examples of structured P2P systems are CAN [13], Chord [14], Pastry [15], and Tapestry [16]. In the next two sections, we study this two P2P approaches in detail.

2.3.1.1 Unstructured P2P System Gnutella

Gnutella is one of the real P2P networks. Gnutella network consists of peers connected via TCP/IP connection which dynamically change. Each peer acts as a client (query source), a server (query destination) and as a router (transmitting of queries and responses). They provide client-side interfaces through which users can issue queries and

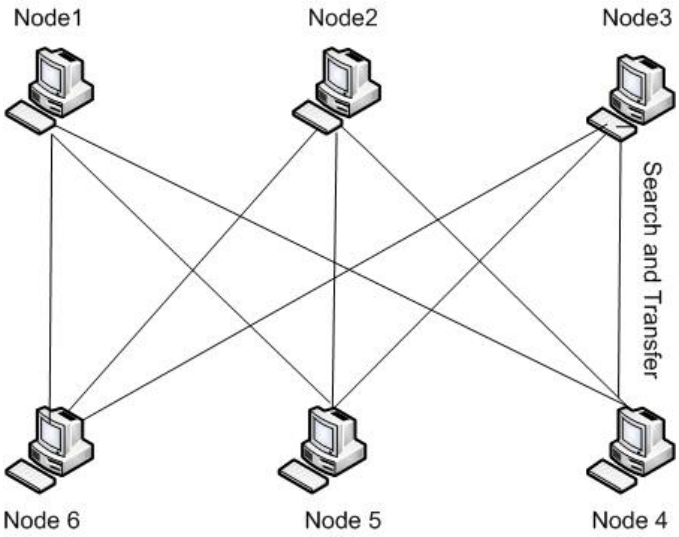

Figure 2.2: Decentralized Peer-to-Peer Network

view search results. They accept queries from other Gnutella nodes (servents), check the matches against their local files and respond with the corresponding results. In this system, users join the network and randomly set-up connection to some nodes as peers. Since the peers are selected at random, they form an unstructured overlay network. To locate a file, a peer initiates a controlled flooding of the network by sending a request packet that contains the file name or a keyword to all its neighbours. Upon receipt of a request packet, a peer reviews its local files to see if it matches the query. If it matches, the peer sends a query response packet back to the sender requests on the reverse path. Fig. 2.2 explains this architecture.

To help maintain the overlay as users enter and leave the system, the Gnutella protocol uses ping-pong messages that help peers to discover other nodes. Pings and pongs behaviour are similar to the query and response messages: all the peers who see ping message send a pong message back to the query source and forward the ping message to its own

set of neighbours. Ping packets flood over the network. Apparently, if the request message travels too far, it could overload the network. The range of flooding is controlled with a time-to-live (TTL) field which is decremented at each hop. Peers sometime create new neighbour connections with other peers that discovered through the mechanism of ping / pong. Each request message also has a universally unique identifier. When a node receives a request message, it checks the ID of the message. If the node has received this message before, it will drop the message. Otherwise, it continues to spread the message to its neighbours. A disadvantage of Gnutella and other similar systems is TTL segmentation. The TTL segments the Gnutella network into sub-networks, which require each user a virtual ”horizon” beyond which their message cannot reach [17]. However, if the TTL is deleted, the messages are exponentially reproduced over the network.

To overcome these problems, the third generation of P2P (e.g. FastTrack, KaZaA and current Gnutella clients) uses a combination of the central server and fully decentralized framework. In this hybrid model, some network nodes are elected as ”super-nodes” or ”Super-Peers” and act as traffic officers for other nodes. The super-nodes dynamically change as there is a change in the bandwidth and network topology.

As shown in Fig. 2.3, nodes with sufficient resources (bandwidth, memory and CPU power) are automatically elected as Super-Peers. A client node maintains number of open connections to these super nodes. In this way a network can scale easier by reducing the number of nodes involved in message handling and routing, as well as reducing the actual volume of traffic between them. When a node wants to locate a file, it sends a query to its Super-Peer. If the Super-Peer finds the index file locally, it sends a response to the node. Otherwise, it forwards or broadcast the query to other Super-Peers. This architecture is also

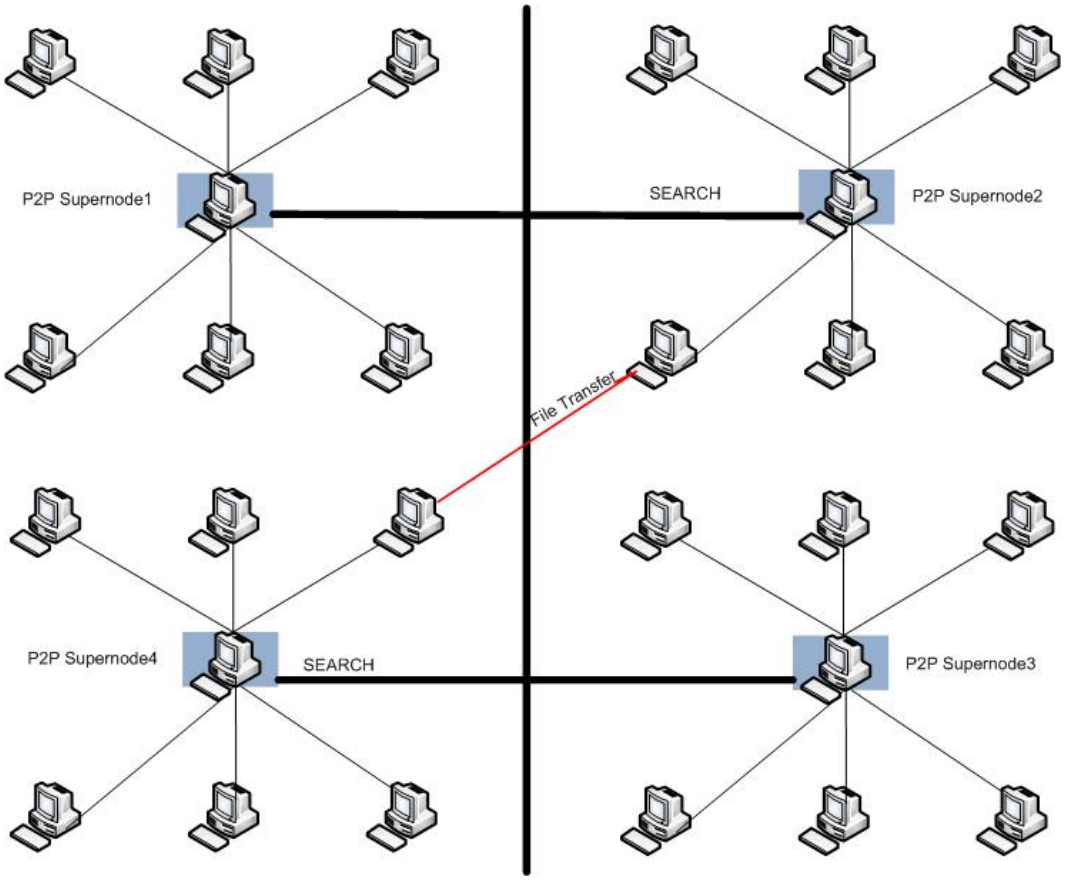

Figure 2.3: Partially decentralized unstructured P2P system

called as a partially decentralized unstructured P2P system, while the original Gnutella network is classified as purely decentralized unstructured P2P system.

2.3.1.2 Structured P2P System Chord

Chord is a structured peer-to-peer network protocol which uses a ring topology. In structured P2P networks, the overlay network is tightly controlled, and the files are placed at specific locations for effective query search (Fig. 2.4). Each host maintains a distributed routing table (DRT) that is used to route requests to the host which has the file [18]. DRT

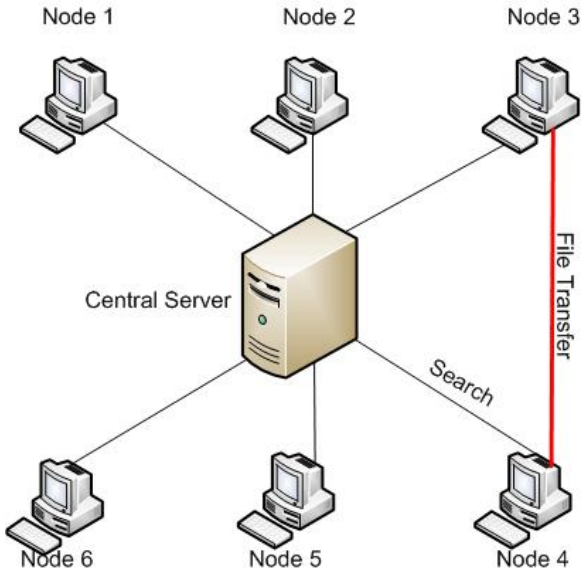

Figure 2.4: Centralized Peer-to-Peer Network

provides mapping between the identifier of the file (for example, a file reference identifier) and the location where the file is stored (for example, the IP address of a host). The basic chord is very simple and describes how nodes join the ring, how data is stored and how the ring recovers from node failures. As shown in Fig. 2.5 Chord provides only one function: given a key, it maps the key to a node. This node will generally be responsible for storing the data associated with that key, or it could store information about where the data can be found.

How does the Chord protocol works?

1. **Consistent Hashing**

   A consistent hash function, such as SHA-1 [14], is used to generate an *m-bit* node identifier and an *m*-bit key-identifier. The node identifier is generated by hashing the node address (i.e. its port and IP address), while the key identifier is obtained

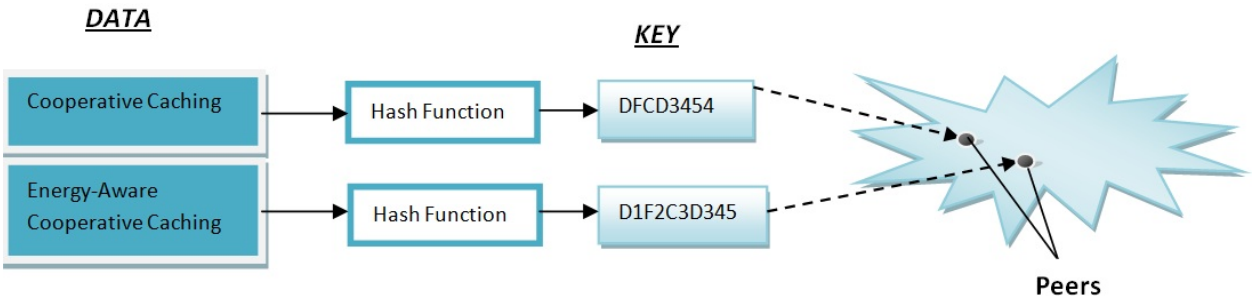

Figure 2.5: Chord Hashing function

by hashing the data that is supposed to be stored in the ring. The identifier length m should be chosen large enough to make the probability of hashing two node addresses or keys to the same value negligible. The node identifiers are arranged in a circle module $2m$. This circle is called the Chord ring (Fig. 2.6). Every key k is assigned to the first node whose identifier n is equal to or larger than k. This node is called the successor node of key k. In a circular representation with identifiers increasing clockwise, keys are assigned to the first node that lies clockwise to them.

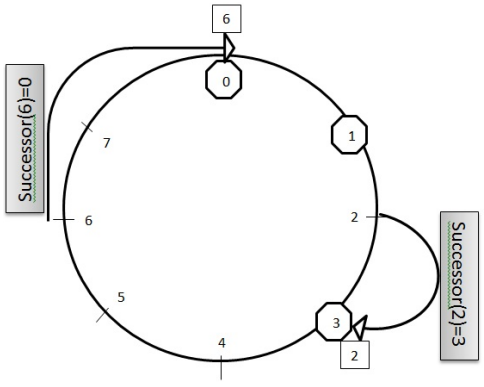

Figure 2.6: An identifier circle consisting of three nodes 0, 1 and 3. Key 6 is assigned to node 0 and key 2 to node 3

2. **Scalable key location**

   Finding the node that matches a key in the ring is easy and needs almost no routing information. If each node knows its immediate successor node, queries can be routed through the ring by a simple transition from one node to the next node. Finally, if the application reaches a node *n* with $n \geq k$ (i.e. the node that succeeds key *K*) is the node that query maps to. However using this method is very inefficient, especially in large rings. In such a scenario, an application might require traversing the entire ring until it finally gets the node corresponding to the query. To increase the speed of applications, Chord maintains additional routing information. This additional information is not essential for correctness, which is achieved as long as the successor information is maintained correctly [14].

3. **Stabilization**

   Keeping successor's pointers up to date is sufficient to guarantee correctness of lookups. These pointers are used to check and correct finger tables, which allow quick and correct lookup. A stabilizing system guarantees adding nodes to the Chord ring while maintaining accessibility to existing nodes in the ring. This must be true even if many nodes join and leave simultaneously. As such, the stabilization will not repair chord ring that has divided to one or more disjoint cycle or cycles. Stabilize periodically run on each node and helps to correct wrong successor which occur due to joining and leaving nodes.

4. **Failures and Replication**

   When a node *n* fails, all nodes whose finger tables include *n* must find the successor

of *n*. Moreover, the failure of *n* must not interrupt queries that are in progress until the system is re-stabilized. As pointed out above, the most important thing in a Chord ring is to keep the pointers correct. To do so, every node keeps list of its *r* nearest successors on the ring. If node *n* notices that its successor **s** has failed, it removes **s** from its list of successors and attempts to stabilize with the next live entry in the list of successor.

Chord is fully distributed, with no node being more important than another. In case of node failures, the system is still able to respond to queries. Even in a continuously changing system, Chord will be able to find the node which is responsible for a key in a defined number of hops.

2.3.1.3 Comparison of Unstructured and Structured P2P Systems

Structured and unstructured P2P systems have advantages and disadvantages. In a purely unstructured system such as Gnutella, essentially nothing should be done to maintain the overlay network Because of the behavior of P2P users, unstructured system is a better way to adapt to the dynamism of P2P systems. However, because of the architecture of unstructured Gnutella, no guarantee can be given that the content can be found. In addition, the messages are coded in plain text and all queries must be flooded in the network. The result is a significant overhead signalling in the network. On the other hand, structured systems provide very efficient technology for forwarding the requests. By applying a predefined hash function to the desired content, the requesting node can essentially determine which node can provide a match for this request before sending the message. Therefore, no real search is necessary. As a request is sent to a specific destination and the destination

node would provide a response to the source node. However, this approach is not easy to support keyword search, which is a very important feature in P2P file sharing. Moreover, it is not as adaptive as unstructured systems when users join and leave. It is interesting to note that neither structured systems nor unstructured consider routing efficiency and network energy consumption as important factors in the construction of the overlay. This is due to the fact that P2P file sharing uses the overlay to direct the queries rather than content, where queries are comparatively much smaller in size and need less energy to transfer. Content is transferred through a separate connection outside the network overlay. Thus, the problem of routing and energy efficiency is not an important factor in P2P file sharing systems. However, these factors may be important for applications such as multi-casting and telephony systems. The effectiveness of routing and robustness can be greatly affects the performance of these applications because the data is transferred over P2P links.

2.4 Power Management Algorithms

Today's mobile devices are growing in number and computational resources. Virtually all mobile devices, laptops, handheld PDAs, smart-phones, even MP3 players have some type of wireless connectivity which make them an ideal platform for peer-to-peer content delivery and data sharing. Each device is capable of storing Megabytes or Gigabytes of digital content. These wireless devices provide a good platform for deploying wireless P2P applications such as video streaming over wireless, content-sharing and data storage. However, always on mode communication patterns of P2P networks are not a natural fit for energy-constrained mobile devices. While hardware advances have led to improved

computational and storage resources, these devices are still severely constrained in energy. Peer-to-peer applications are likely to make this energy shortage worse. Unlike existing client-server applications, where the client can perform disconnected operations and utilize intermittent connectivity, P2P applications assume that peers are always on and available to route messages or satisfy queries for data.

Most research in energy conservation strategies has targeted wireless networks that are structured around base stations and centralized servers. Such networks do not have the limitations associated with small, portable devices. By contrast, an ad-hoc network is a group of mobile, wireless hosts which cooperatively form a network independently of any fixed infrastructure.

There are many power management techniques and algorithms being proposed. Masako Shinohara and Hideki Hayashi [19] proposed a data access disconnection method to balance the power consumption among mobile hosts. In their method, a path between the requested node and replying node (which has data) will be chosen by considering the path length and the remaining amount of batteries of mobile hosts along the path. According to their method, when a mobile host requests a data item, the request immediately will answer if it holds the requested data item by itself. Otherwise, it will be answered through one of its connected mobile hosts (replica). If none of connected mobile hosts hold the data item, the request fails. In their method, they didn't consider any restriction on the replica allocation methods. The multi-hop routing problem in ad-hoc networks has been studied in terms of bandwidth utilization and energy consumption by Huaping Shen and Mohan Kumar [20]. They presented an energy-efficient cache replacement strategy called GreedyDual Least Utility (GDLU). It is a novel energy and bandwidth efficient data

caching mechanism, which enhances dynamic data availability while maintaining consistency. The proposed utility-based caching mechanism considers connection-disconnection, mobility handoff, data update and user request patterns, to achieve significant energy savings in mobile devices. Based on the utility function derived from an analytical model, GDLU selected data items with the most utility to cache in local memory in order to minimize the energy cost at mobile terminals.

Energy-aware design and evaluation of network protocols for ad-hoc networking environment requires practical knowledge of the energy consumption behavior of actual wireless devices. The wireless LAN MAC layer specifications in the current IEEE 802.11 standard support a Power Save Mode (PSM) [21]. PSM conserves energy on idle nodes, by powering their wireless interface off for selected periods of time. Such a Power Save mode is applicable to both infrastructure networks, and ad-hoc networks. In the specification of 802.11 PSM, each station may be in one of two power modes: awake (when the node is fully powered) and doze (when the node is not able to transmit or receive, and consumes very little power). One of the shortages of these methods is the current IEEE 802.11 implementations could not provide any delay-performance guarantee because of their fixed wakeup intervals.

2.5 Summary

In this chapter, we reviewed ad-hoc cooperative caching and ad-hoc caching policies that are commonly used and focused on Hybrid-Cache policy, which is the policy selected for this thesis. We had a brief overview on RAON algorithm, which considers energy and

delay of nodes before forwarding packet to those nodes. We then provided an overview of several existing P2P systems used for file sharing applications and compared their differences. At the end, we discussed some energy conservation algorithms.

Chapter 3

Resource-Aware Cooperative Caching Design

The project's goal is to design a Resource-Aware Cooperative Caching P2P system that is suitable for deployment on MANETs. Due to the constraints and dynamism of MANET nodes, the P2P system must be aware of the available resources and adapt to changes. Among the existing P2P systems, Gia is the one which provides the features that ad-hoc networks can take advantage of. We decided to use Gia as the basis, and extend it to address the problems that exist in ad-hoc network. We have also chosen to use reactive routing, specifically AODV[22], as the underlying routing protocol. This chapter explains the motivation of AODV and Gia. It is accompanied by a detailed description of our mobile ad-hoc P2P system, Resource-Aware Cooperative Caching Overlay Network.

3.1 AODV

Ad hoc On-Demand Distance Vector (AODV) Routing is a routing protocol for MANETs. AODV is an efficient, simple and effective routing protocol. It is a reactive routing protocol, meaning that it establishes a route to a destination only on demand. This algorithm was

inspired by the limited available bandwidth for the media that are used for wireless communications. It borrows most of the useful concepts from DSR and DSDV routing algorithms. The on demand route discovery and route maintenance from DSR and hop-by-hop routing from DSDV make the algorithm cope up with topology and routing information.

In AODV when a node needs a connection in the network, it broadcasts a connection request. Other AODV nodes forward this request, and record the node that they heard it from to create a temporary route back to the requesting node. When a node receives the connection request message and already has a route to the desired node, it sends a message backwards through a temporary route to the requesting node. The requesting node then begins using the route with the smallest number of hops. The features of AODV that make it desirable for MANETs with limited bandwidth include:

- Less complexity: The algorithm makes sure that the nodes which are not in the active path do not maintain information about this route.
- Maximum bandwidth utilization: AODV does not require periodic global advertisements; the bandwidth demand is less.
- Simple: It is a simple protocol because nodes behaving as a router and maintaining a simple routing table. On the other hand the source node initiating path discovery request and making the network self-starting.
- Highly Scalable: The algorithm is highly scalable because of the minimum space complexity and broadcasts avoided compare to DSDV.

Because of the above specifications of AODV, we decided to use AODV as the routing protocol for our project.

3.2 A Scalable Gnutella-like P2P System Gia

Gia, which is an unstructured P2P system, was proposed to improve the scalability and robustness of unstructured network. This section discusses problems in unstructured P2P networks that Gia addresses, followed by the outline design of Gia.

3.2.1 Whys and Wherefores Gia

As described in the previous chapter structured P2P systems use DHT to organize the overlay and routed messages between nodes. . In particular, the hash function defines the specific nodes that may store objects as well as nodes which participate in lookups for specific object. Thus, any file can be simply located by sending a request to the node that has the file pointer. The file pointer provides a mapping between the file and its location.

Although this technique is much more scalable than unstructured P2P systems, but it only supports exact match queries. The exact name of a file is translated into a search key and the corresponding look up key. However, DHT is less able to support keyword searches: given a sequence of keywords, it searches the corresponding files. The current use of P2P file sharing systems requires such keyword matching. Conversely, this method is very costly in terms of maintenance because nodes join and leave frequently [23]. Another disadvantage of structured systems is maintenance of the structure. It is difficult to maintain the structure which is needed to efficiently route messages in a very transient network. As discussed earlier, the high rate of joining and leaving in a transient P2P network causes a significant overhead for the system to maintain a stable condition. An unexpected failure of a node, without informing its neighbors first, requires more time and work into DHT to

(a) discover the failure and (b) replicate lost data or pointers. After the failure is detected, the file pointers of those previously stored file in the failed node need to be recovered and transferred to another node or nodes.

However, these issues do not affect systems such as unstructured Gnutella. Therefore, a new P2P file sharing system, Gia, which is built on the Gnutella design, is proposed in [24].

3.2.2 Gia Design

Gnutella-like systems have a problem: when they face with high collections of query rate, nodes quickly become overloaded and the system stops functioning satisfactorily. Moreover, this problem is compounded when the system size increases. Furthermore, the scalability of Gnutella is a critical issue because of the flooding mechanism used in the search protocol (Fig. 3.1). The concept of SuperNodes used in FastTrack technology and the latest version of Gnutella, helps improve the scalability of Gnutella networks. Recently, instead of broadcasting queries to all peers, the idea of random walk is using [25]. Even though, this method greatly reduces the number of requests generated at each node, it is essentially a blind search which may increase search time significantly.

Gia modified Gnutella to integrate the ideas of random walk and Super-Node. The primary objective in designing Gia was to create a Gnutella P2P type system capable of handling much higher rate of queries. The second objective was to make Gia more scalable and continue to work well as the system sizes increases. To achieve the scalability, Gia tries to avoid overloading the nodes by explicitly take into account constraints of node capacity.

The Gia design consists of four main components, i) active flow control scheme, ii) search protocol, iii) dynamic topology adaptation and iv) one-hop replication of file point-

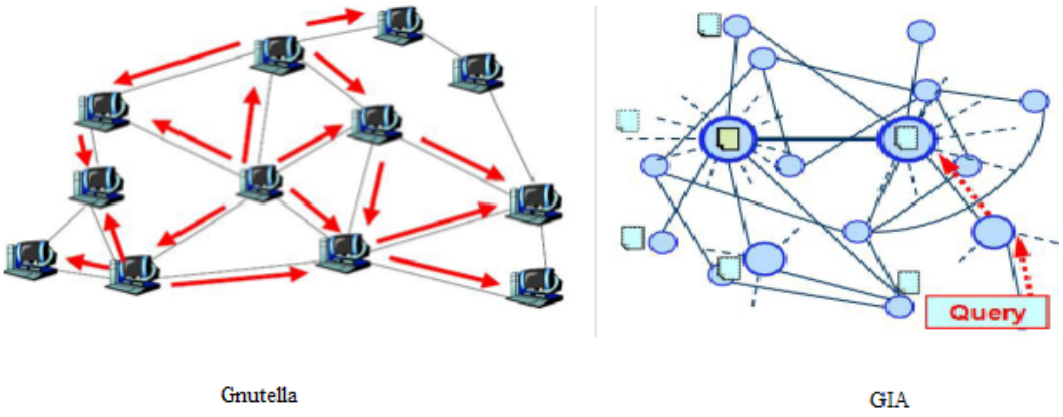

Figure 3.1: Flooding in search method of Gnutella. Searches based on biased random walks in GIA

ers.

The need for flow control has been recognized as a weakness in the original design of Gnutella. Gia introduces an algorithm based on token flow control to avoid overloading the hot-spots. The flow control protocol acknowledges the existence of heterogeneity. This protocol adapts to the changes by assigning flow control tokens to nodes based on their available capacity. Each node has a pool of tokens and distributes these tokens to its neighbours according to the node capacity from time to time. These tokens represent the amount of request messages that a node is ready to accept from its neighbours. Each time a node sends a request to a neighbour it consumes a token that it received from the neighbour. If a node uses all tokens of a particular neighbour it cannot forward the requests to that node until it receives a new set of tokens. The token allocation algorithm is based on Start-time Fair Queuing (SFQ), where the capacity of the node will be considered as weight. Tokens that were assigned to inactive neighbours are automatically redistributed proportionally among the other neighbours. As neighbours join and leave the tokens would accordingly be distributed.

Gia's unique design achieves a comprehensive system capacity. Gia replaces Gnutella's flooding with random walks where if a node has valid tokens from a neighbour with high capacity it forwards query to that neighbour (Fig. 3.1). This protocol is based on the perception that high capacity nodes can often provide useful answers to many queries. Each query is assigned a globally unique identifier (GUID). If a node receives a query that has received it before, it would forward it to a different neighbour. $Maxresponses$ and *TTL* are associated with each query parameters. The *TTL* value is decremented each time the request is forwarded, and $max\_responses$ is decremented when a query hit is sent. When one of these values reaches zero, query will be dropped. As a result they define how far a query can travel.

Gia recognizes the propositions of overlay network topology while using random walk and thus includes a topology adaptation algorithm. The topology adaptation algorithm is used for the structure of the overlay topology which ensures to place most of the nodes in short range of high capacity nodes. The adaptation protocol makes sure that high degree nodes, which receive most of the requests, actually have the ability to process these requests. Each node is assigned a capacity level as the number of requests which the node can handle per second. The number of connections from one node to other nodes depends on its capacity level. High capacity node makes more connections. Neighbours inform each other of their capacity levels and the current level when the connection is initially established. They also send periodic update messages to inform each other about changes in their level. Each node independently calculates a level of satisfaction (*S*), which is a value between 0 and 1 representing the degree of satisfaction from its neighbours. A value of 0 means it is not satisfied at all and it would look for new neighbours, while a value of

1 means that it is fully satisfied and no need to make new connections. This value is calculated by adding the capacity of all the neighbours (normalized to their degrees) divided by the node's capacity. This ensures that high capacity nodes have more neighbours than low-capacity nodes, where they can act as Super-Nodes. This number is also used to determine the range of adaptation, which dictates the aggressiveness of the topology adaptation algorithm.

To establish new connections, Gia node (say *X*) identifies other existing peers first. Node discovery can be achieved by contacting a bootstrap server to use ping-pong messages. After a list of existing peers is defined, *X* maintains the list of peers in its cache, and chooses a candidate from the list, (say *Y*) with the highest capacity and attempts to connect to it. When *Y* receives a connection request, it would automatically accept the connection if it has not reached to its maximum number of neighbours (each node has a $maxnbr$ parameter that is determined by its capacity). Otherwise it would compare the capacity of *X* with the capacity of its existing neighbours. If choosing *X* as a new neighbour can help *Y* to be more satisfied, then *Y* would accept the request and drop one of its neighbours.

Gia also implements a one-hop replication, where each node maintains an index of the contents of all its neighbours. According to the topology adaptation algorithm, high capacity nodes tend to have more neighbours which cause to contain a larger list of index file. It means that the one-hop replication ensures nodes with high capacity are able to provide answers to a larger number of requests.

While Gia does build on these previous contributions, Gia, to our knowledge is the first open design that (a) combines all these elements and (b) considers the peers' capacity constraints and adapts its protocols to reflect these constraints.

In summary, Gia is a system that takes advantage of the benefits of unstructured systems, and improves scalability at the same time by controlling how the network is built. Due to the topology adaptation protocol and one-hop replication, Gia is capable of using a biased random walk, to provide a higher success rate in the search query. In a research at the University of Maryland, they compared the performance of multiple search methods for existing unstructured P2P. The results show that Gia is a very good all-around solution, combining different ideas from other schemes [26]. In addition, the topology adaptation protocol and flow control mechanism makes the P2P topology and query forwarding to be aware of the nodes' capacity. Since the resources are very limited in the MANET environment, we believe that the design of Gia might fit well in it, hence we decide to use Gia as the basis for our project.

3.3 Resource-Aware Overlay Network

Mobile devices and ad-hoc network will be universal in a not too distant future. Thus, it is important that any P2P overlay network should perfectly work with physical underlay networks, including wire-line, wireless cellular and ad-hoc networks. For this seamless integration however the overlays should account for, apart from the traditional challenges of P2P like decentralization, self-organization and unreliable peer availability leading to topology dynamics, the peculiarities of the underlying mobile ad-hoc network, particularly resource constrains like memory of portable devices, bandwidth, power, low computation capability, unpredictable (dis)connectivity and dynamics of topology because of peer [27].

Most of the current P2P systems are mainly designed for stationary hosts that have

reliable network connectivity and virtually unlimited energy supply. Therefore, we need to re-evaluate these systems under the new environment (wireless networks).

Structured P2P systems, unlike the first generation of P2P which uses flooding, are much more effective. But to achieve the desirable efficiency, these systems impose the structure and thus may become a risk to the dynamics of the network. Structured P2P systems require each node to hold indexes for any node in the network, even though the index could be pointing to a node that is away from you, both virtually and physically.

If a node leaves the P2P network, its index files must be maintained by another node. This operation is expensive. When a node crashes without warning, the recovery operation is even more expensive because the neighbouring nodes must first detect the failure and then recover the pointers which were stored in the failed node. In MANET, network connections can be immediately disconnected due to the nodes' movement and limited power. This poses a huge problem for structured P2P systems since failures can frequently occur. Therefore, it is important to achieve not only efficiency but also to ensure the reliability of such systems. Thus, we consider the unstructured P2P systems to be a better solution than structured. Of all the unstructured P2P architectures that explored, we believe that the design of Gia might fit well in the ad-hoc environment.

In this thesis, we design and evaluate Resource Aware Cooperative Caching techniques to efficiently support data access in mobile ad-hoc networks. We essentially use Gia as the foundation, and add the features to address problems specific to mobile ad-hoc networks. In this section, we first discuss some of the problems that Gia might face when running on ad-hoc network, followed by a detailed description of our network.

3.3.1 Gia's Adaptability to Ad-hoc Networks

The abstraction of capacity at a node plays an important role in the four design features of Gia (topology adaptation, flow control, one-hop replication, and biased random walk). Topology adaptation protocol puts most of the nodes in short range of high capacity nodes. The adaptation protocol ensures that high degree nodes, which receive most of the requests, actually have the ability to process these requests. The flow control protocol explicitly acknowledges the heterogeneity existence and adapts to it by assigning flow control tokens to nodes. Tokens assignment is based on the available capacity of the nodes and also to limit the number of queries a node can send to the other. In fact the topology adaptation algorithm guarantees relationship between high capacity nodes and high degree nodes. One-hop replication ensures that high capacity nodes are able to provide answers to a larger number of requests. Finally, biased random walk increases the probability of query hit by forwarding queries to the high capacity nodes, which are generally better able to respond to requests.

Node capacity in Gia, is defined as the number of requests it can handle within a time period. This value is mainly determined by the speed of CPU, memory and bandwidth of the node, which are usually constant for a static wired node. In contrast, Ad hoc nodes have the characteristics of mobility, unstable connection, and limited power. These factors may affect both the node capacity and network performance.

The biased random walk algorithm selects the neighbour with highest capacity for which it has flow-control tokens and sends the request to that neighbour. This method significantly reduces the number of query hit rather than a pure random walk. Gia uses *TTL* to limit the biased random walk interval and book-keeping techniques to avoid redundant

paths. With book-keeping, the node assigns a unique GUID to each request. As a result, the node remembers the neighbours which it has already sent requests for a given GUID. If a request with the same GUID arrives at the node, it sends it to a different neighbour. This reduces the probability that a query traverses the same path twice. This approach is ideal for ad-hoc network, because it reduces the number of messages generated per query.

On the other hand, since Gia was not specifically designed for wireless networks, it does not take into account the limited resources of wireless nodes. In ad-hoc environment capacity is no longer a constant parameter and changes based on the state of the underlying network. Since the biased random walk always have a tendency to transfer requests to the node with high capacity, links to these nodes would experience more traffic than other links. If any of these links are unstable, it would affect the performance of any application that passes through the high capacity nodes. The performance of a search query is defined as the time needed for the query source to receive a query hit response for a particular query. Unlike flooding, where many copies of the same query is generated for each search, only one copy is generated by biased random walk. In addition, a high capacity node usually processes more messages than others, leading to expend energy at a higher rate.

Taking into consideration of the power constraint, our suggested algorithm modifies Gia's design to adapt to this constraint in ad-hoc networks.

In this thesis we design a Resource-Aware Cooperative Caching system which is based on Gia and takes into account residual energy of nodes, demand and supply of ad-hoc network to distribute the file among the nodes.

3.4 Resource-Aware Cooperative Caching

Most researches in ad-hoc networks focus on routing and not much work has been done on data access. A common technique used to improve the data access performance is caching. Cooperative caching, which allows sharing and coordination of cached data among multiple nodes, can further explore the potential of the caching techniques. Due to mobility and resource constraints of ad-hoc networks, cooperative caching designed techniques for wired network may not be applicable to these networks. In this thesis, we design and evaluate cooperative caching technique to efficiently support data access in ad-hoc networks and also reduce the network's energy consumption. As Liangzhong Yin and Guohong Cao proposed in [7], CachePath and CacheData can significantly improve system performance. The analysis showed that CachePath performs better when the cache is small or the data update rate is low, while CacheData performs better in other situations.

In order to reducing the network's energy consumption, we propose a caching algorithm which checks network's resources such as residual energy (RE), demand and supply of network to make caching decision.

3.4.1 Proposed basic Cooperative Cache Scheme

In ad-hoc networks, a data request is forwarded hop-by-hop until it reaches to the data center (or a node which already cached data). Then, the data center sends the requested data back. Various routing algorithms have been designed to route messages in mobile ad-hoc networks. To reduce the bandwidth, energy usage and the query delay, the number of hops between the data center (or caching node) and the requester node should be as small

as possible.

Gia and RAON, use biased random walk to forward queries. The critical difference between Gia and RAON is that the capacity level of a neighbour is the only factor that preferences the random walk in Gia. Our model likewise RAON consider the nodes' energy to publish and forward a query for data. On the other hand, in the proposed algorithm, query source make decision where to fetch data. Each intermediate node which receives the request and has cache-path information forwards its information to the query source by using point query. In this way each query node has a better understanding of the network's graph. As a result, we can increase the network performance by considering the nodes' energy to download from and reducing the network's energy consumption.

Within this system, each intermediate node keeps track of sources of the request and responding nodes for data d_i with the purpose of knowing demand and supply of the network and make caching decision individually. In such a network, when a node which has data responds to a request, it also adds its residual energy, demand and supply which it receive for the data d_i and the expiry time (t_e) of the cached object to the respond message. Therefore the intermediate nodes can keep track of resources in network.

Furthermore, the battery size usually varies from one device to another. Therefore, we decided to use energy states instead of the actual value. Each node monitors its neighbours energy level and determines their energy state relative to their battery and cache size. We define energy states of HIGH, MEDIUM, and LOW.

3.4.2 System Model

Fig. 3.2 shows part of an ad-hoc network. Some of the nodes in the mobile ad-hoc network may have wireless interfaces to connect to the wireless infrastructure such as wireless

LAN or cellular networks. File server contains database for d_i. Note that file server, which has the database, may be a node connecting to the wired network.

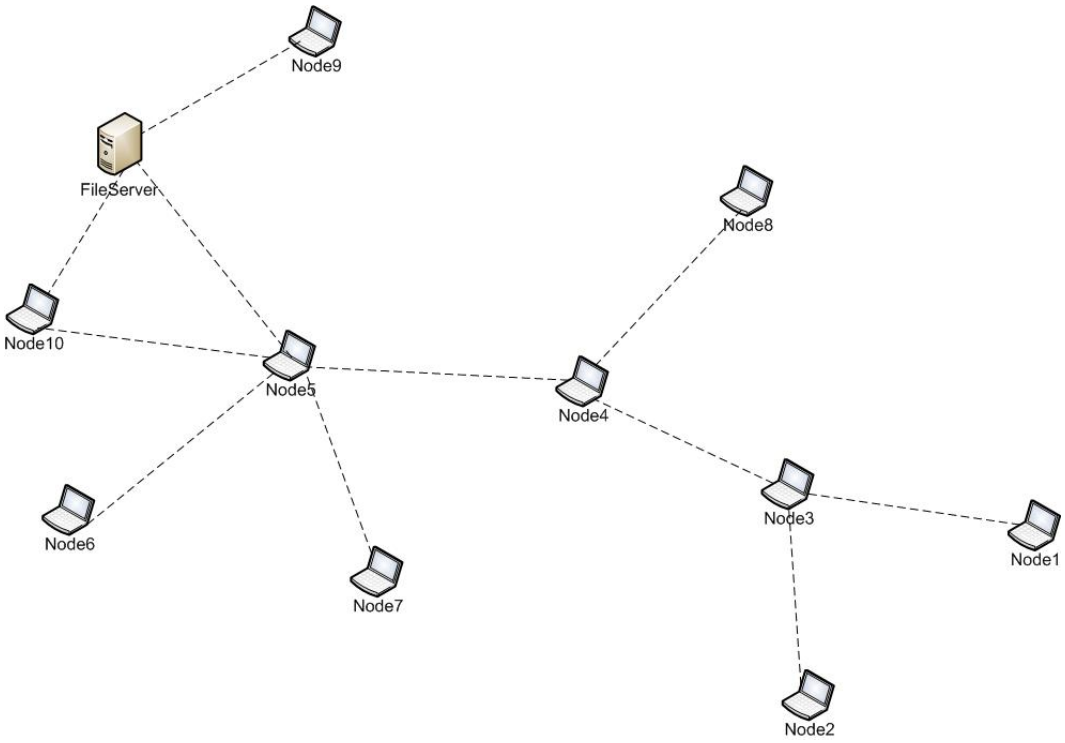

Figure 3.2: Ad hoc network

In this network when a node wants to send a request, it finds it's neighbors. Suppose node X wants to find its neighbors in the network, it broadcasts ***Check−Neighbor*** message to find it's directly connected neighbors. When node Y receives this request, it finds a route to the requesting node by sending ***RREQ−AODV−Message***. Node X replies back to this request by publishing ***Route−Reply*** message.When node Y receives the reply message, it sends acknowledgement to node X (***WLAN−ACK***). Y also replies back to the neighbor's request from node X by **Neighbor−Reply** message. Node X confirms the route and they become neighbors. Fig. 3.3 shows these sequences.

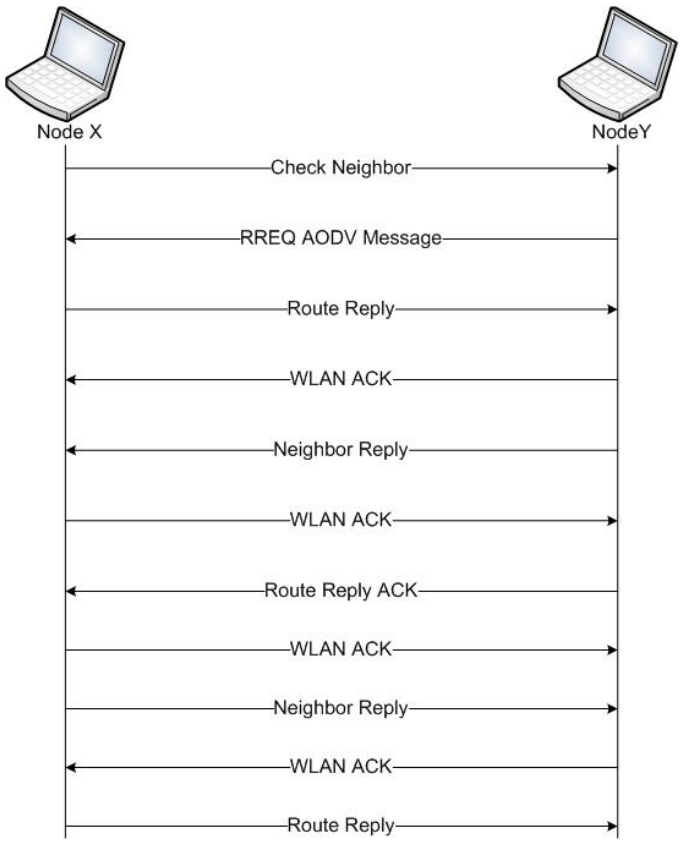

Figure 3.3: Neighbor Discovery

As illustrated in Fig. 3.2 if node N_1 requests data item d_i from file server, this request first reaches N_3 (intermediate node). N_3 checks its cache to see if it already cached d_i or has a cached-path for d_i. If so, it responds back to the requesting node with the list of suppliers for the data therefore the source of the query can make decision where to fetch data (Fig. 3.4). If it doesn't see any information for d_i, it forwards the request to the next hop.

When N_3 receives the respond packets for object d_i, it processes to find a candidate as the best supplier(s) in order to cache-path. N_3 calculates the residual energy of each supplier per request, to see how many requests it can handle. Then it chooses a node with

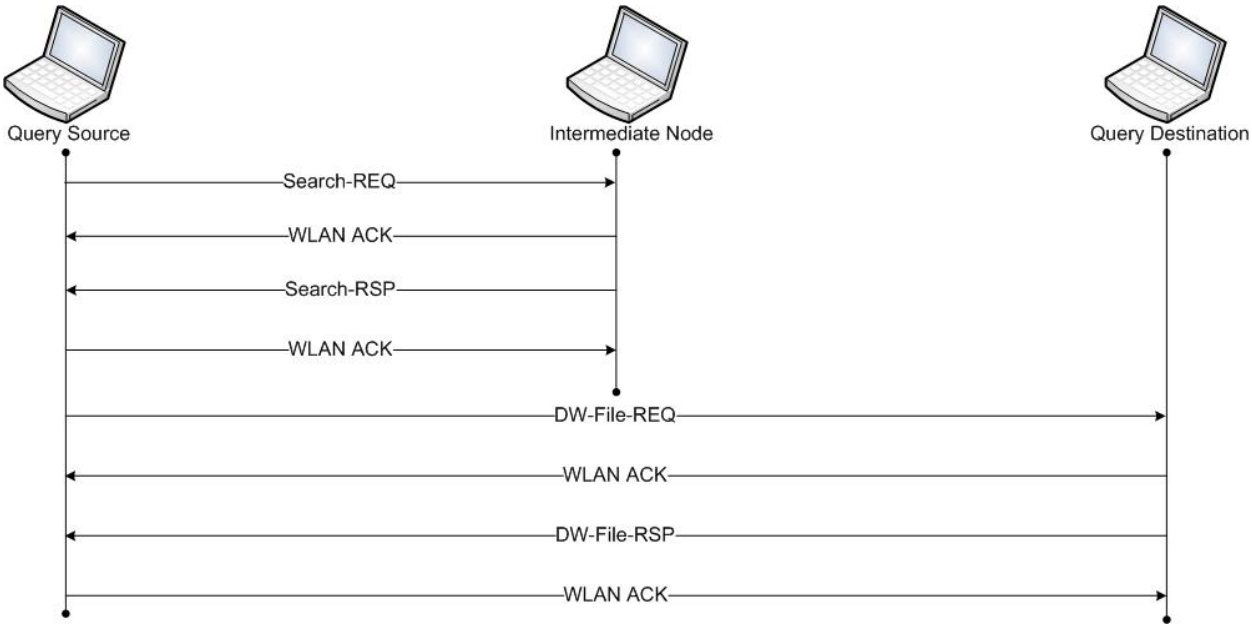

Figure 3.4: File Search

the higher residual energy in order to answer more requests in future and it keeps Node-Id(s) of successor(s) and next hop to reach this (these) supplier(s).

3.4.3 RACC Approach

RACC is a hybrid scheme which exploits the strengths of CacheData and CachePath while avoiding their weaknesses. Specifically, when a mobile node forwards a query reply for a data item or forwards data to the query source, it caches the data or path to the node which has data. In the following sections, we describe RACC approach and the parameters that it considers to make cache-path or cache-data decisions and compose our algorithm.

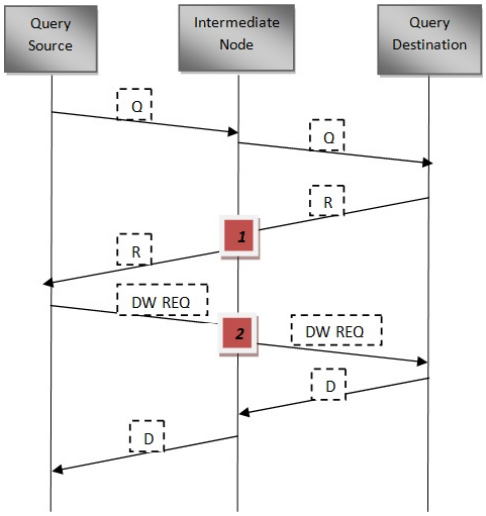

Figure 3.5: Caching Decision. It will make decision cache path (*1*) ,It will make decision to cache data(*2*)

3.4.4 Caching Decision Sequence

Once an intermediate node receives query for object d_i, it keeps track of the query and forwards it to the next hop. As soon as there is a query hit, the respond message comes back to the requested node. As shown in Fig. 3.5, the intermediate node makes caching decision in 2 different sequence; it makes Cache-Path decision when it sees reply packet for requested data and makes Cache-Data decision when it receives download data request.

3.4.4.1 Caching the Data Path (Cache-Path)

In Cache-Path, mobile nodes cache the data path and use it to reference future requests to the nearby node which has the data instead of the faraway file server. As we mentioned it before, when a node which has data (supplier) answers a query, it adds also its RE and the expiry time (t_e) of the cached data to the respond packet. Since shown in Fig. 3.2, when N_3

receives the respond packet for object d_i, it checks it's table to find a candidate as the best supplier to cache its path and its NodeID in order to refer future requests to that supplier. Fig. 3.6 shows this sequence. An intermediate node receives 3 queries for data d_i (Q_1,Q_2 and Q_3). It passes these queries to its neighbors. When it receives reply for object d_i, it caches the path to the query destination. If another query comes to this intermediate node (Q_4), this node replies back to the query source with the list of suppliers which it cached their path before.

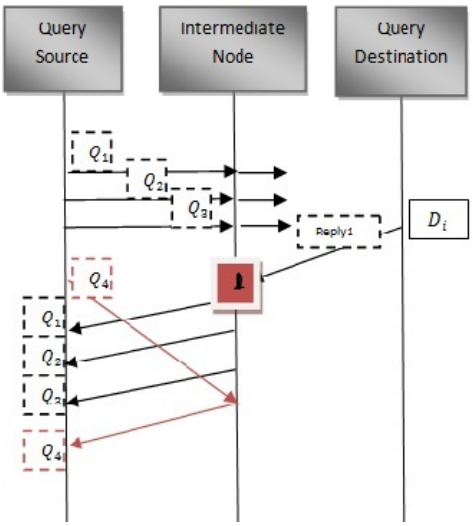

Figure 3.6: Example of Cache-Path Sequence

N_3 checks residual energy of each supplier and assign RE state to them (HIGH, MEDIUM, LOW). Then, it checks download path cost per request to see how many requests it can handle. As result, a node with higher RE and lower DW path cost will be chose. N_3 caches the successor NodeID and path for supplier to refer future request to that supplier until the next selection. Fig. **??**.B shows this algorithm that applies these heuristics in RACC.

3.4.4.2 Caching the Data (Cache-Data)

In Cache-Data, intermediate nodes cache the data to serve future requests instead of fetching data from other nodes. In this scheme, the node caches a passing-by data item d_i locally when it finds that d_i is popular (i.e. there were many requests for d_i), its residual energy is HIGH and it has enough free cache space. When N_3 receives a download request for data item d_i, it makes caching decision either to cache or not to cache the data. Fig. **??**.C shows the algorithm that applies these heuristics in RACC.

3.4.4.3 Demand and Supply Estimation of the Network

In the network, query can be generated in different parts of the network. To make a proper caching decision on each intermediate node, we suggest considering another important factor: the demand and supply of the network. In order to have better view of demand and supply in the network, it is better also considering queries from different paths than the path which intermediate node is located. An intermediate node *i* in our model keeps track of demand and supply. They use these records to 1: make caching decision. 2: provide useful information for query source in order to have better view of the graph of the network. By following so, the performance of the network can be improved by reducing the delay.

1. **Demand Estimation**

   In our system we have *n* query sources $\{S_1,S_2,...,S_n\}$ and *j* query destinations $\{D_1,D_2,...,D_j\}$. Estimation of demand at t_c for data d_i can be done by calculating the number of query sources which each intermediate node reports while it forwards the query packet.

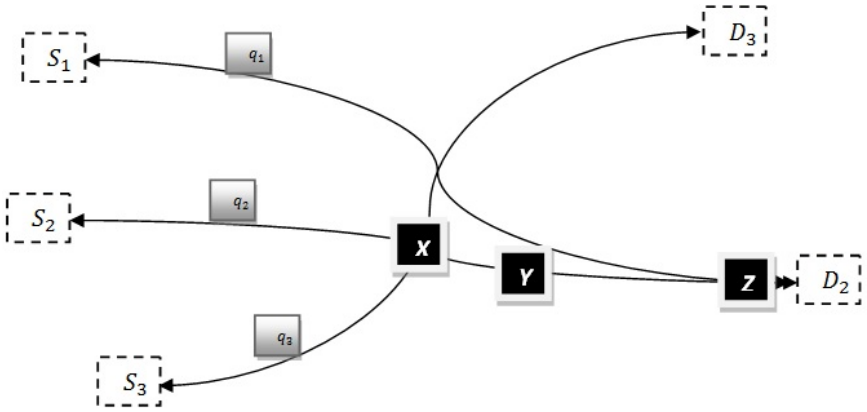

Figure 3.7: Query Source 1, Query Source 2 and Query Source 3 send query for data that is located in Destination*2* and Destination*3*.

When intermediate nodes receive a query packet for data, each node also puts other query sources for this data that it hears and forwards the query to the next hop. When query destination receives this packet, it calculates the sum of the demand and puts this information to the query responds packet.

If we assume $\{u_1,...,u_n\}$ represents different numbers in the range of $\{1,2,...,n\}$ and $\{v_1,...,v_j\}$ refers to different numbers in the range of $\{1, 2, ..., j\}$, then as showed below, query destination can estimate the demand and inform other intermediate node about the demand.

Query Request: $\{(S_{u_1},D_{v_1}), (S_{u_2}, D_{v_2}),...,(S_{u_n}, D_{v_j})\}$

Query Respond: $\{demand = n\}$

By following this method, all the intermediate nodes are aware of the current demand in the network.

To illustrate the above finding, we give an example.

Fig. 3.7 shows a case in our model where *X*, *Y* and *Z* are 3 different types of intermediates nodes. *X* is an intermediate node which is located in the intersection of 2 query paths (S_2, D_2 and S_3, D_3) . *Y* is an intermediate node which lays on a single query path (S_2, D_2) and *Z* is an intermediate node which is located in a sub-path of 2 queries (S_2, D_2 and S_1, D_2).

Worth mentioning that there is a relationship between time and the number of queries that each node may hear from the paths which it is laid on. As an instance, in the above scenario *Y* may or may not know information about both S_2 and S_3. It depends upon the time that S_2 and S_3 publish their request.(Fig. 3.8)

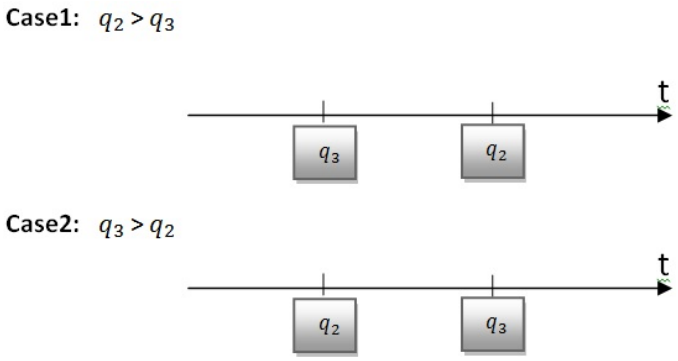

Figure 3.8: Time relationship between Query sent time and the number of Demand

(a) Case1: S_3 sends query before S_2

When *X* receives the query packet from S_2, it will import all of the query sources that is aware of (S_2, S_3) to the packet and forwards it to *Y*. Moreover *Y* knows that for the data d_i there is 2 demand at time t_c which is S_2 and S_3.

(b) Case2: S_2 sends query before S_3

When *X* receives the query packet from S_2 , it is not aware of S_3 and it adds only S_2 to the query packet and forwards it to *Y*. Thus *Y* doesn't know anything about S_3 at time t_c until it receives next query packet for the data from *X* later.

2. **Supply Estimation**

In our network we have 3 types of suppliers. *n* Query sources, *j* Query destinations and *m* Potential caches. Query sources are the nodes which transmit the request for data (S_1,S_2,...,S_n). Query responders are either servers or the nodes that cached data before (D_1, D_2,...,D_j). Potential cache, are those intermediate nodes having the potential to cache data (HIGH residual energy) and they make caching decision whether to cache or not to cache data ($Node_1$, $Node_2$,...,$Node_m$).

When a query source receives a query response, it sends download request packet. In the reverse path, interested intermediate node in caching data, adds its Node-Id into the packet and forwards the packet. As download request packet reaches to the query destination, it copies the Node-Id of the interested intermediate node(s) into the reply packet and sends back the data packet to the requested node(s).

If S_n sends a download request for data d_i to query destination D_j and k intermediate nodes which having the node-Id of $\{k_1,k_2,..,k_m\}$ decide to cache the data, then as demonstrate here, query destination use this information to estimate supplier and also inform intermediate nodes about this estimation.

DW Request:$\{Node_{k_1}, Node_{K_2},..., Node_{k_m}\}$

DW Reply: $\{D_j, Node_{k_1}, Node_{K_2},..., Node_{k_m}, S_n\}$

3. **Demand and Supply Equation**

   Fig. 3.9 illustrates the demand and supply equation. Upon reception of a query packet the node checks it to see if all the neighbour query sources are included. If they are not, the node adds these sources into the packet and sends the packet to the next neighbor. When the query destination receives this query packet, it calculates the sum of the demands and puts the number of demands in the reply packet. Then it sends the packet toward the reverse path to the query source. Consequently the query destination informs the intermediate nodes about the demand in the network. For supply calculation, an intermediate node which receives download request packet, checks if it is interested in caching data. If it is interested, it adds its Node-Id into the packet and forwards it. When Query destination receives this packet it copies the Node-Id of the interested intermediate nodes into the reply packet and sends back the data packet to the requested nodes. Therefore nodes in the reverse path can have the estimation of the number of suppliers in the network.

Table 3.1: An Example of Demand and Supply Table.

Data d_i	Query-Sources Node id	Successor Suppliers Node-Id	RE of the suppliers	Next-hop neighbor to supplier	The path to supplier (# of Hops)	Expiry time(t_e)
d_1	S2	D2	100 W	Z	3	30
d_1	S3	D3	100 W	X	3	30

Intermediate nodes check Query respond and DW reply packets to keep track of demand and supply for data d_i and build up their demand and supply table for this data. When they want to make caching decision for the data d_i, they check the number of demand and supplier for this data in the network. If the number of demander is more than supplier, they

cache the data. Otherwise they just cache the path to the data. Table.3.1, shows an example of demand and supply table for data d_1.

3.5 Proposed Algorithm

As described in this chapter the goal of our scheme is to distribute data among the nodes according to their energy states by considering demand and supply of the network. According to RACC approach:

- When an intermediate node receives a *Query* for data d_i: It checks if it has any cache-path or cache-data information for this data. Therefore it can directly answer the query
- When an intermediate node receives a *Query Respond* for data d_i: It checks which suppliers it can cache their paths for this data (cache-path).
- When an intermediate node receives *Download Request* for data d_i: This node checks if it can cache the data (cache-data).
- When an intermediate node receives *Data item* d_i: If this node requested data, it caches d_i. Otherwise it forwards data to the next hop.

Fig. 3.10 shows the work flow of RACC.

```
handleMessage(message)
{
        If (message == Query && message->Destination =! thisNode->Address)
        {
                vector<IPAddress> DemandArray;
                DemandArray = CheckQuesryData(message->dataId);
                for(int i; i<= DemandArray.size();i++)
                        If(DemandArray[i] != message->DemandArray[i])
                                message->DemandArray.push_back(DemandArray(i);
                UseRandomWalk(message);
        }
        else if(message == Query && message->Destination == thisNode->Address)
        {
                ResponseMessage Response = new ResponseMessage();
                Response->DemandsCount(message-> DemandArray.size());
                Response->setType(QueryResponse);
                //.
                // fill other Parameters of response message
                //.
                sendResponseMessage(Response);
        }
        else if(message == DWRequestingData && message->Destination =! thisNode->Address)
        {
                if(ThisNode->RE == HIGH)
                        Message->CapableSupplier.push_back(ThisNodeAddress);|
                ForwardUseAODV(message);
        }
        else if(message == DWRequestingData && message->Destination == thisNode->Address)
        {
                ResponseMessage Response = new ResponseMessage();
                Response->SupplierCount(message-> CapableSupplier.size()+2); // 2 = source and destintation are counted
                Response->Data(myfiles.getFile(message->dataId));
                Response->setType(DWResponse);
                //.
                // fill other Parameters of response message
                //.
                sendResponseMessage(Response);
        }
        else if(message == QueryResponse)
        {
                if(message->Destination == thisNode->Address)
                        {
                                DataInfo dInfo = new DataInfo();
                                dInfo.Demand = message->getDemandsCount();
                                dInfo.Data = message->data;
                                dInfo.supplier = 0;
                                dInfo.DataDemanders = message->getDemandArray();
                                DataList.Add(dInfo);
                        }
                        else
                                ForwardUseAODV(message);
        }
        else if(message == DWResponse)
        {
                        if(message->Destination == thisNode->Address)
                        {
                                updateDataSupplier(message->dataId,message->getSupplierCount());
                        }
                        else
                                ForwardUseAODV(message);
        }
        else
                delete;
}

vector<IPAddress> CheckQuesryData(int dataId)
{
        vector<IPAddress> demandArray;
        For(int i ; i<= ForwardedPackets.size() ; i++)
                If(ForwardedPackets[i]->info.dataId == dataId)
                        demandArray.push_back(ForwardedPackets[i]->info.Address);
        return DemandArray;
```

Figure 3.9: Demand and Supply calculation in RACC

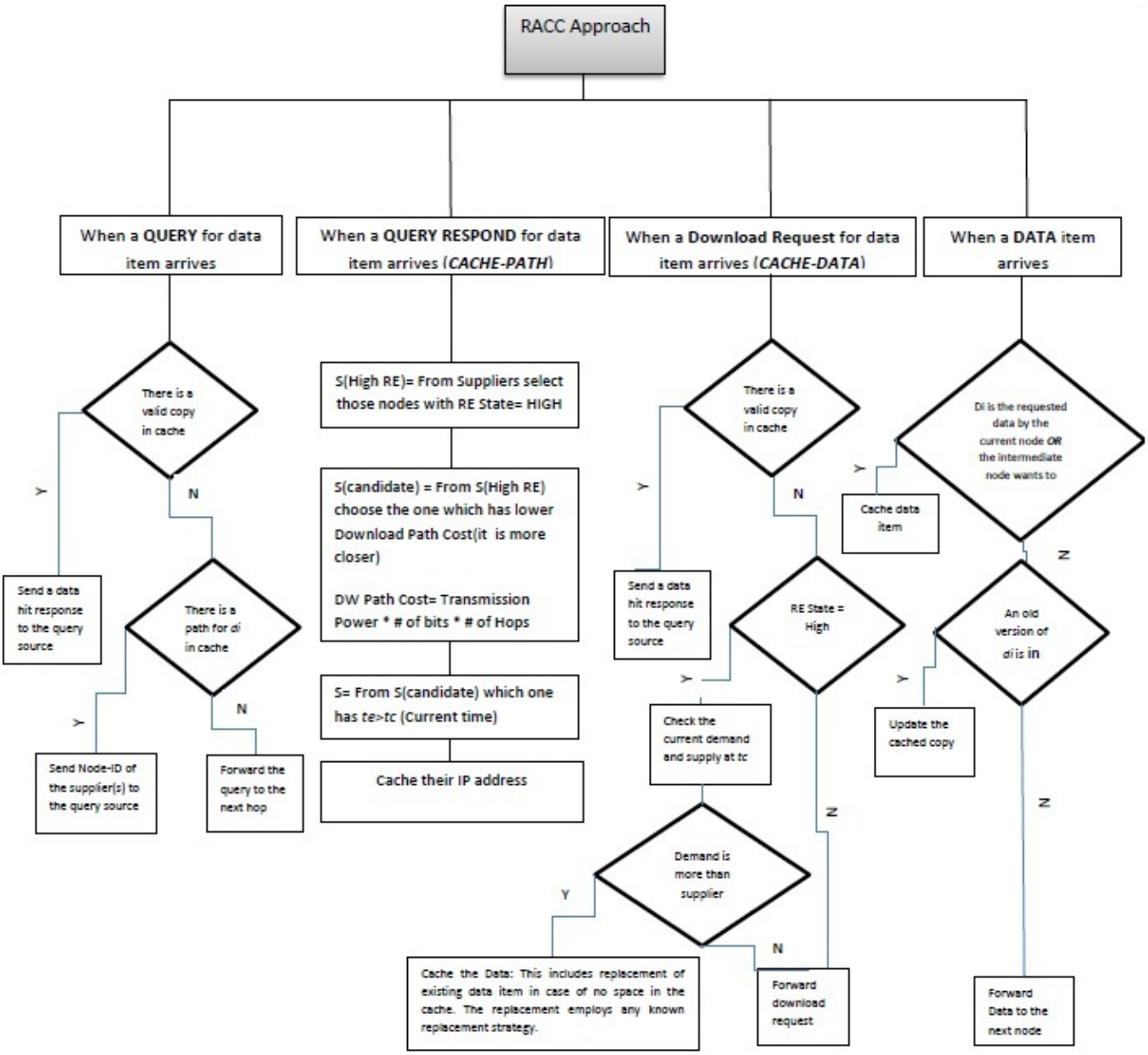

Figure 3.10: RACC Workflow

Chapter 4

Evaluation

We implemented RACC algorithm described in Chapter 3, Simple Cache, CachePath and CacheData using OMNET++. In this chapter, we describe our simulation environment and the simulation set-up. We then provide the obtained results, along with analysis and comparison.

4.1 Simulation Environment

Objective **M**odular **N**etwork **T**est-bed in **C++**(**OMNeT++**)[28] is a public-source, component-based, modular simulation framework which is developed at the Technical University of Budapest, Department of Telecommunications (BME-HIT). OMNeT ++ is a discrete event simulation environment. It's primary application area is the simulation of communication networks. It provides component architecture for models. Components (modules) are programmed in C++, and then assembled into larger components and models using a high-level language (NED).

OverSim [29] is an open-source overlay and peer-to-peer network simulation framework for the OMNeT++ simulation environment. The simulator contains several models

for structured (e.g. Chord, Kademlia, Pastry) and unstructured (e.g. GIA) P2P systems and overlay protocols. OverSim was developed at the Institute of Telematics (research group Prof. Zitterbart), Karlsruhe Institute of Technology (KIT) within the scope of the ScaleNet project.

OMNeT++ does not come with any modules to simulate TCP/IP, wireless and mobile simulations networks. This is provided by the INET and INETMANET framework. The INET Framework [30] is an open-source communication networks simulation package for the OMNeT++ simulation environment. The INET Framework contains models for several wired and wireless networking protocols, including UDP, TCP, SCTP, IP, IPv6, Ethernet, PPP, 802.11, MPLS, OSPF, and many others.

Support for mobility and wireless communication has been derived from the INETMANET [31]. This Module is based on INET Framework and continuously developed. Generally it provides the same functionality as the INET Framework, but contains additional protocols and components that are especially useful while modeling wireless communication. The core framework implements the support for node mobility, dynamic connection management and a wireless channel model. The framework can be used for simulating: several routing protocols(e.g. AODV), fixed wireless networks, mobile wireless networks, distributed (d-hoc) and centralized networks, sensor networks, multichannel wireless networks.

OMNeT++ version 4.1 together with OverSim 20101103 and INETMANET have been chosen as the simulation platform. The main reasons include the ease of use, a highly flexible and modular architecture as well as an open-source code base. OMNeT++, OverSim and INETMANET are built as hierarchical architectures. Thus, at the lowest level of the

hierarchy simpler modules encapsulate the desired behaviour. These modules are implemented as C++ classes. More complex modules (compound modules) may be composed of several simpler modules as well as other compound modules. The communication among modules is attained through message-passing.

4.2 Simulation Set-Up

We implement a cross-layer design of a mobile ad-hoc network. When an intermediate node receives a packet (not route discovery packets), it sends the packet up to the application layer in order to make caching decision. We use the IEEE 802.11g standard MAC layer. We exploit "Ieee80211NicAdhoc" module from INETMANET framework as the network card for all the wireless nodes. Energy consumption in mobile nodes can be categorized into 2 different sets: Processing energy consumption and Communication energy consumption. In our simulation, when a node is processing an event the average CPU usage is 7.6% which needs 3.3mA (11.88 W) of energy. If the node is in idle state, the CPU usage is 0.237% which consumes 0.237 mA(0.85 W) of energy. For communication power, each node has 3 operation modes during communication with its neighbours; sending, receiving and idle mode. When a mobile node is in transmission mode the energy consumption is 15.6mA(56.16 W), when a node is receiving a packet energy usage is 19.47mA(70.092 W) and when node is in idle mode the usage is 0.37mA(1.332 W).

Transmission range in our network is set to 150m. The INET battery module in INET framework is used to model the nodes energy consumption. In our simulation each intermediate node assigns 3 states of energy to its neighbours; HIGH, LOW and Medium.

Node movements are modeled with the random waypoint mobility model, where each node chooses a destination randomly within the simulation area. 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 moves on to another destination. Table.4.1 and Table.4.2 show the channel physical and NIC parameter which we used in our simulation.

Table 4.1: channel physical parameters

Parameters	Values
*.channelcontrol.carrierFrequency	2.4GHz
*.channelcontrol.pMax	2.0mW
*.channelcontrol.sat	-110dBm
*.channelcontrol.alpha	2
*.channelcontrol.numChannels	1

Table 4.2: nic settings

Parameters	Values
**.wlan.mgmt.frameCapacity	10
**.wlan.mac.address	”auto”
**.wlan.mac.maxQueueSize	14
**.wlan.mac.rtsThresholdBytes	3000B
**.wlan.mac.bitrate	2Mbps
**.wlan.mac.retryLimit	7
**.wlan.mac.cwMinBroadcast	7
**.wlan.mac.cwMinBroadcast	31
**.wlan.radio.bitrate	2Mbps
**.wlan.radio.transmitterPower	2mW
**.wlan.radio.thermalNoise	-110dBm
**.wlan.radio.sensitivity	-85mW
**.wlan.radio.pathLossAlpha	2
**.wlan.radio.snirThreshold	4dB
**.*wlan.radio.phyOpMode	”802.11g”

Our simulation objective is to study the behavior of Simple Cache, CacheData, CachePath and Resource-Aware Cooperative Cache(RACC) and evaluate their performances under

different system configurations. Simple cache is a non cooperative caching technique in which nodes only cache data if they requested that data, otherwise nodes don't collaborate in caching data.

By varying the maximum speed of mobile nodes, we generated mobile ad-hoc network scenarios with various levels of dynamism. Table.4.3 shows the network level simulation parameters. The maximum speeds of 2m/s, 5m/s, and 20m/s are used to model the movements of pedestrians, non-motorized vehicles, and motor vehicles respectively. Simulation area is 1000m x 1000m. We simulated behavior of Simple Cache, CacheData, CachePath and RACC as changes happen on the number and speed of nodes.

Table 4.3: Network level simulation parameters.

Parameters	Values
Simulation Period	250 sec
Simulation Area	1000m x 1000m
Maximum Speed	2m/s, 5m/s, 20m/s
Pause-Time	30 sec

The simulation begins with a defined mobile ad-hoc Network topology where the nodes are randomly placed over the simulation area. We used AODV as the underlying routing protocol in MANET [22]. At the P2P level, each overlay node uses a uniform random number between [0.5s, 1.5s] to determine when to join the network. Therefore, no P2P topology is defined in the beginning of the simulation. A Tracker or Bootstrap node exists in the simulation which assigns nodes configuration like Mac and IP Address that is vary from node to node. After a node is connected to the overlay network, it periodically sends a message to find its neighbors every *update-interval* time. The *update-interval* in our model is 10 sec. In this model, all nodes, except the file server, are mobile nodes. 100 files are located in the file server. The file size is 100 KB to 300 KB. When the nodes join

the network, their cache is empty. Each node sends a request for the file that it doesn't have in its cache. Queries are generated at a random rate between [10s, 30s] to model the behaviour of aggressive P2P users. Each file has a unique file-Id for which nodes send a request. The cache sizes of nodes are different. This is to reflect that different wireless devices have different levels of resources. The cache size is assumed infinite so that cache replacement policy has no impact on the RACC performance. In this model when a query source generates a request for a specific file and receives the file, it will keep file until the end of simulation. But there is a time expiry for the data which is cached on the intermediate nodes. Intermediate nodes cache data for 30 seconds. Meanwhile, if an intermediate node receives another request for the same cached file within these 30 seconds, its expiry time will be reset. Table.4.4 summarizes all the simulation parameters used at the P2P level.

Table 4.4: P2P level simulation parameters.

Parameters	Values
Number of peers	10, 20, 30, 40, 50
Query Generation Interval	Uniform random between [10, 30]
Number of files stored at server	100
Update-interval	30 sec
Query-timeout	5 sec
File Size	[100kb-300kb]
Data Expiry time(t_e)	30sec
Transmission Rate	2Mbps
TTL	30

4.3 Simulation Results

To measure the impact of RACC on mobile ad-hoc cooperative caching system, we look at three system performance descriptors: average query delay, number of hop counts and average RE. We compare our proposed Resource-Aware caching scheme with Simple

Cache, CacheData and CachePath schemes when number and speed of the nodes change.

- **Average Query Delay**

  The average query delay is defined as the time interval between the moment when a data query is sent and the moment when the data frame is received from the server or an intermediate node. The average query delay has been measured for different number of nodes with different speeds. Fig. 4.1, Fig. 4.2 and Fig. 4.3 show increasing in number of nodes increases the demands for data. As a result, more nodes compete for limited bandwidth causing average query delay growth for all four schemes. However, the average query delay increase rate of the proposed scheme is smaller than that of other mechanisms. This can be explained by the fact that if Simple Cache cannot find data frame in its cache, it sends out the request to the server which triggers to a long delay. CacheData and CachePath take into consideration only the neighbors' cache to improve the delay of network. However, RACC experiences less delay by utilizing the strength of CachePath and CacheData while considering demand and supply of the network.

  From Fig. 4.1, Fig. 4.2 and Fig. 4.3 we understand:

  1. When the number of nodes is small, RACC performance improvement is not magnificent compared to CacheData. When the number of nodes increases, RACC outperforms considerably other the three schemes.
  2. By increasing the mobility of the nodes, the collision probability in the network increases which triggers larger packet retransmission time. The average query delay of CachePath scheme is larger when the mobility of nodes increases due

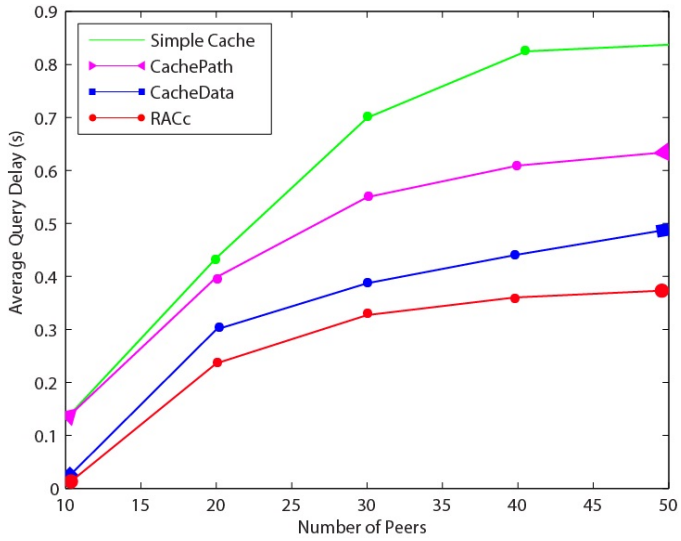

Figure 4.1: Average Query Delay(Speed=2m/s)

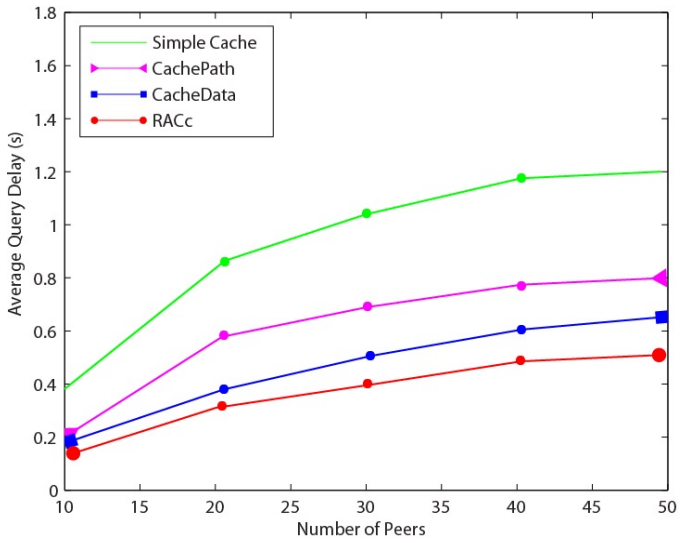

Figure 4.2: Average Query Delay(Speed=5m/s)

to the more frequent path changes. However, RACC experiences less query delay as the speed increases compared to the other three schemes.

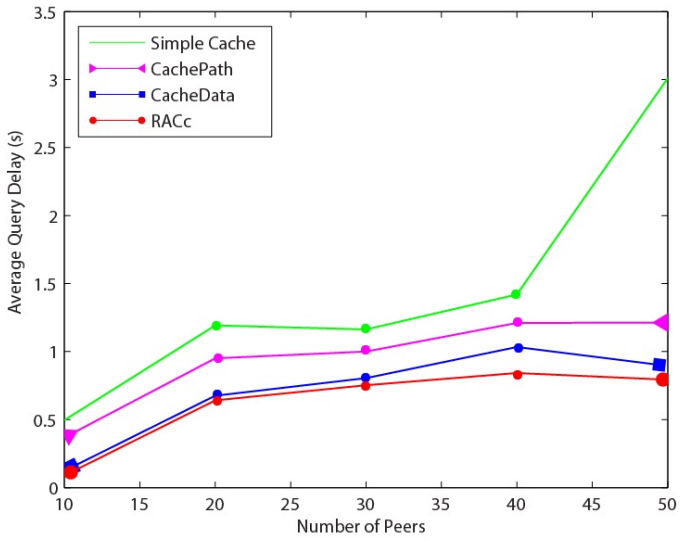

Figure 4.3: Average Query Delay(Speed=20m/s)

- **Average Hop Count**

  The hop count refers to the number of hops in a path between the demander and the supplier. AODV provides the hop count information between the source and destination of the query. To reduce energy usage and the query delay, the number of hops between the query source and the query destination should be as small as possible. In the proposed system, each intermediate node tracks the number of the hops between itself and the suppliers in order to find nearest supplier having HIGH residual energy. Decreasing the hop counts degrades the packet transmission energy.

  Fig. 4.4, Fig. 4.5 and Fig. 4.6 plot the average hop count between source and destination. The average query delay is approximately equal for different speeds. It is mainly affected by the node density of the network. Graphs show that the average hop count between supplier and demander in RACC are approximately equal to 2

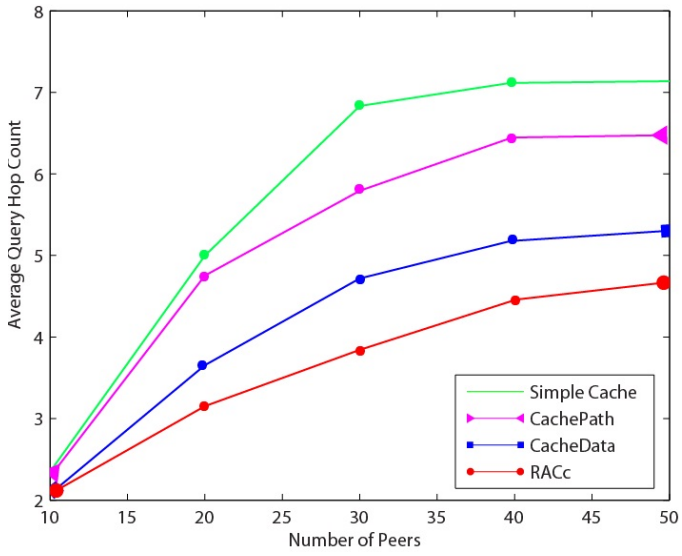

Figure 4.4: Average Query Hop count (Speed=2m/s)

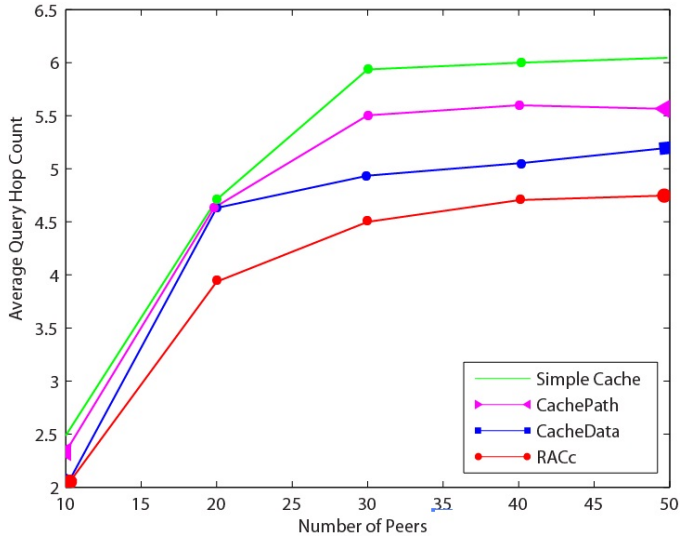

Figure 4.5: Average Query Hop count(Speed=5m/s)

where there are 10 nodes in the network and 4.5 where 50 nodes are in the area.

In Simple Cache, intermediate nodes don't participate in caching. Therefore each requesting node usually gets response from far away data destination. RACC utilizes

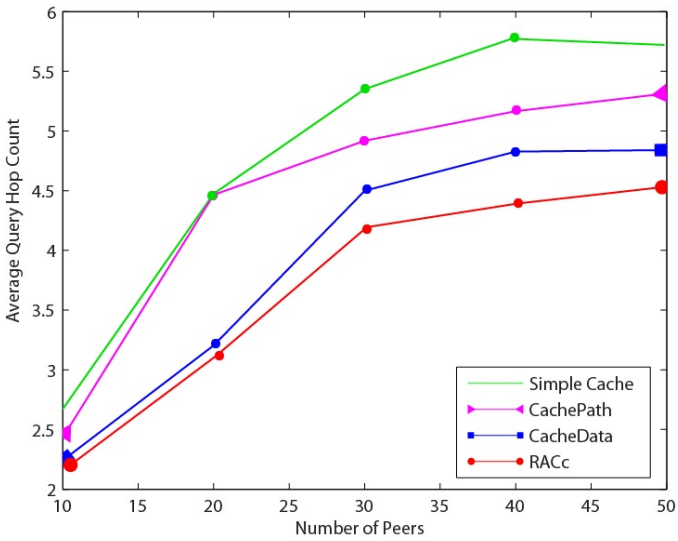

Figure 4.6: Average Query Hop count(Speed=20m/s)

both CachePath and CacheData to improve the average query delay of the network. When an intermediate node has low RE, it only caches the path to suppliers whereas when the energy of node is high enough it caches the data. When an intermediate node, which has either path or data cached, receives a query packet, it forwards all the information related to path or cached data towards the query source.

- **Average Residual Energy**

  RE plays an important role in RACC. Each node makes caching decision by considering residual energy to see how many requests can be responded by this amount of energy.

  In our MANET network, all of the nodes start with the same amount of energy. RACC considers node energy before caching decision making. Thus, the probability of having a dead node in the network during the simulation time is small. The

RE of nodes in the network decreases by almost equal amount. Fig. 4.7, Fig. 4.8 and Fig. 4.9 show the Average RE of the network for different speed and number of nodes.

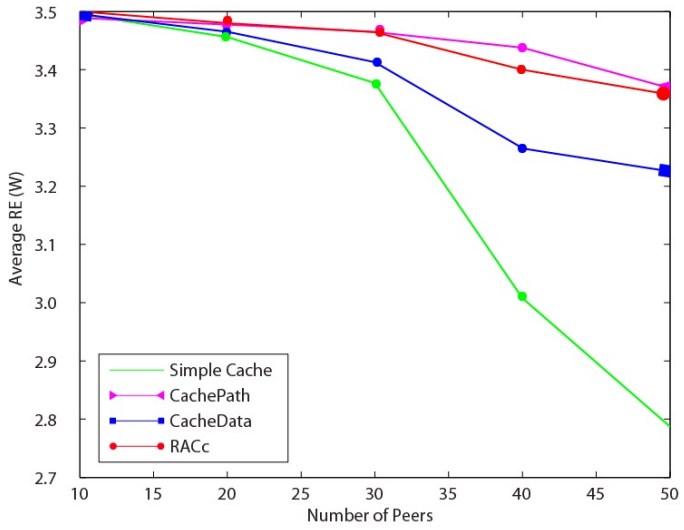

Figure 4.7: Average Residual Energy(W)(Speed=2m/s)

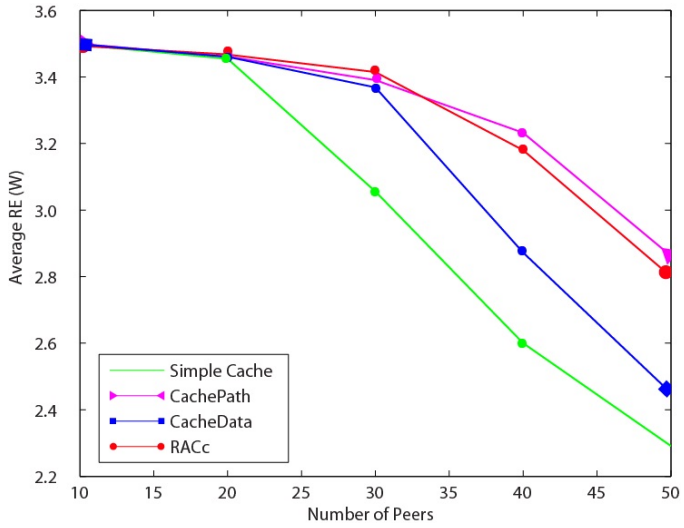

Figure 4.8: Average Residual Energy(W)(Speed=5m/s)

The graphs indicate that RACC outperforms the other three schemes in terms of RE

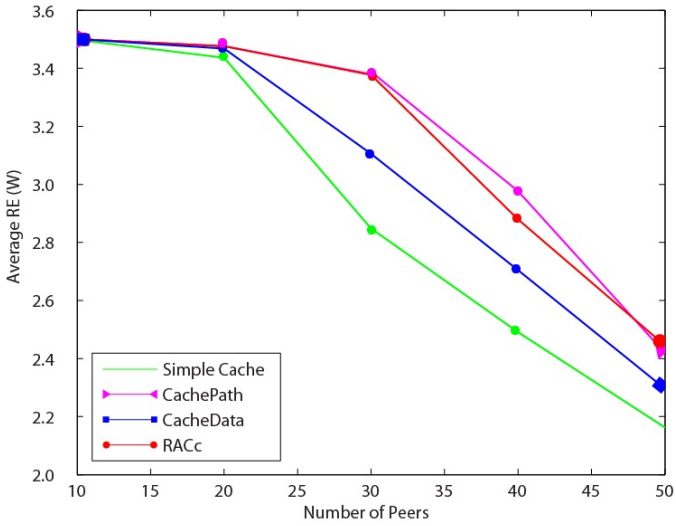

Figure 4.9: Average Residual Energy(W)(Speed=20m/s)

of the network. The graphs show CachePath scheme uses less amount of energy than CacheData and sometimes RACC, since according to CachePath scheme nodes only cache the path not the data, which consumes less energy.

The growths of mobility and number of nodes increase the energy consumption in the network. Thus the energy of the whole network decreases.

In the following findings are described:

1. As the number of nodes increases, nodes forward queries to more nodes which consume more transmission power.
2. In the wireless network, nodes transmit requests to all neighbours in their transmission range. By increasing number of nodes, they receive more packets which are not destined to them(overhear). Overhearing packets also requires energy usage.
3. Air is a shared medium between the nodes in a network. When the number of

nodes increase, the possibility of collision in the network increases which consume energy as well.

- **Cumulative Distribution Function(CDF)**

We plot CDF of residual energy in Fig. 4.10 for the speed equal to 2m/s with 50 peers in Simple Cache, CacheData, CachePath and RACC. As an example, for RACC, when RE is equal to 3.4 W, the CDF graph shows the value of 0.7. It means 70% of the nodes in the network has $RE \leq 3.4$ W.

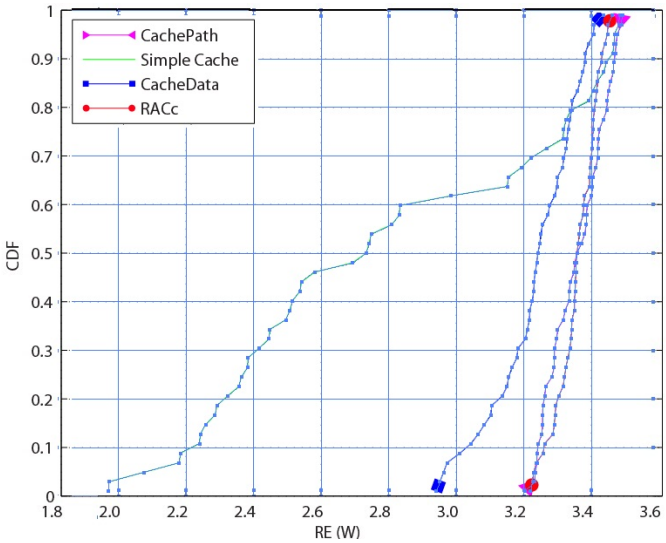

Figure 4.10: CDF of Residual Energy for 2:1000 with 50 peers

We discover that the RE of the nodes by using RACC has smaller variance compared to the other schemes which indicate RACC mechanisms consume less energy. For instance, when we run simulation for 250 seconds, the RE doesn't get smaller than 3.2 W.

It means that by considering Residual energy of each node, demand and supply during caching decision making the network lifetime increases.

Chapter 5

Conclusion and Future work

5.1 Conclusion

P2P and ad-hoc networks have many similarities, including decentralization, self-organization and dynamic topology. A key issue in both systems is the process query content discovery in a decentralized manner. Thus, it is clear that the P2P model is particularly suited for ad-hoc network to allow communication and distribution of content in this environment. Most existing P2P systems designed for the Internet, and dont consider cooperation between P2P and ad-hoc networks. Therefore the P2P overlay network is not aware of the underlying physical network.

Most research in ad-hoc networks focus on routing, and not much work was done on data access. A common technique for improving the performance of data access is caching. Cooperative cache allows sharing of cached data among multiple nodes and explores the potential of caching techniques. Due to resource constraints in ad-hoc networks, cooperative caching techniques designed for wired network may not be applicable to these networks.

In this thesis, we design and evaluate a Resource-Aware Cooperative Caching technique to efficiently support data access in mobile ad-hoc network. Our system makes caching

decisions taking into consideration of the ad-hoc characteristic such as node power constraints. In particular we design caching policy which also considers residual energy of each node as well as demand and supply of the network to make caching decision and distribute data among the nodes. Our design is based on Gia, which is an unstructured P2P system. In support of the caching policy, we consider a hybrid cache policy which exploits the strengths of CacheData and CachePath while avoiding their weaknesses. Specifically, when a node forwards a data item, it caches the data or path based on its residual energy, query destination residual energy as well as demand and supply of the network.

As described before the purpose of this design is to reduce the energy consumption of the network and delay that each node faces in the network. Within such a system when a node receives a packet, adds its RE to the packet and forward it to its neighbours. Intermediate nodes assign a RE state to their neighbours (according to RE that they announce) as HIGH, MEDIUM and LOW. To facilitate caching decision intermediate node checks its energy state and also its neighbour energy state to make the best caching decision in order to reduce delay and energy consumption of the network.

We evaluated the performance of Simple Cache, CacheData, CachePath and RACC using OMNET++ . Simulation results show that RACC improved query delay and energy usage as compared to other schemes under a variety of topology and load conditions. It reduced the delay which each intermediate node may face while sending a query and receiving the data by considering demand and supply of network. Fig. 5.1, shows the average delay improvement of RACC compared to the other 3 schemes. For instance, RACC improves the average delay by 26.9% compared to Cache-data for speed=2m/s. Also it reduced the number of dead node by considering RE of each node while making caching

decision.

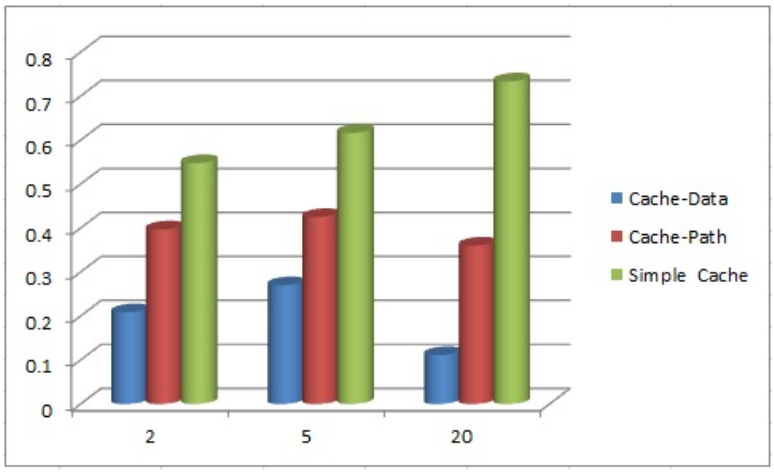

Figure 5.1: Average delay improvement percentage by RACC over other methods for 50 nodes with different speeds(2m/s, 5m/s, 20m/s)

5.2 Suggestions for Future Work

There is a lot of study and progress to make in the area of cooperative caching. These studies suggest a variety of research directions that need to be followed to make such a system feasible.

One such direction would be designs which consider real-time media traffic such as voice and video. Typically these applications have high data rate requirements and severe delay constraints, whereas wireless nodes generally have limited resources in energy and bandwidth. We may consider video streaming as our application in which video is divided to the chunk or segment and we can use our algorithm to make caching decision how we distribute these chunks on the network in order to keep the real-time characteristics of this application and use less amount of energy.

Another possibility would be to study effects of having a directory server which can

track demands and supplies. In our design intermediate node individually make caching decision. The directory server keeps track of demands and supply for data. In such a system when a node wants to send request they will forward the request to the directory server and directory server provide them list of suppliers.

In addition, mobile technology has reached a stage that enables easy access to information technology anywhere, any time. Following our design, Node stationary in ad-hoc networks can be another factor which each intermediate node considers in order to make caching decision. In this way data will be cached in nodes with low mobility rather than the nodes with high mobility and movement.

Finally, considering the impact of limited cache size and cache replacement policy on the overall RACC performance are 2 more factors which is left as our future work.

Bibliography

[1] L. A. Prashant Kumar, Naveen Chauhan and N. Chand, "Enhancing data availability in manets with cooperative caching," *International Journal of Mobile Computing and Multimedia Communications*, vol. 3, no. 4, pp. 53–66, October-December 2011.

[2] A. Rousskov and D. Wessels, "ICP and the squid web cache," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 3, pp. 345–357, April 1998.

[3] (2012) The cache array routing protocol (CARP). [Online]. Available: http://msdn.microsoft.com/en-us/library/ms812590.aspx

[4] G. Cao, L. Yin, and C. Das, "Data replication for improving data accessibility in ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 5, pp. 1515–1532, January 2006.

[5] M. Papadopouli and H. Schulzrinne, "A performance analysis of 7ds a peer-to-peer data dissemination and prefetching tool for mobile users," in *Advances in wired and wireless communications, IEEE Sarnoff Symposium Digest*, March 2001.

[6] U. C. Kozat, ztan Harmanc, S. Kanumuri, M. U. Demircin, and M. R. Civanlar, "Peer assisted video streaming with supply-demand-based cache optimization," *IEEE TRANSACTIONS ON MULTIMEDIA*, vol. 11, pp. 1515–1532, April 2006.

[7] L. Yin and G. Cao, "Supporting cooperative caching in ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 5, pp. 77–89, January 2006.

[8] B. Tang, H. Gupta, and S. Das, "Benefit-based data caching in ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 7, pp. 289–304, January 2008.

[9] J. Zhao, P. Zhang, G. Cao, and C. R. Da, "Cooperative caching in wireless p2p networks: Design, implementation, and evaluation," *IEEE Transactions on Parallel and Distribution System*, vol. 21, pp. 229–241, February 2010.

[10] G. Lau, M. Jaseemuddin, and G. Ravindran, "Raon: A p2p network for manet," in *Second IFIP International Conference on Wireless and Optical Communications Networks (WOCN)*, Toronto, ON, CANADA, 2005, pp. 316 – 322.

[11] (2012) Napster protocol specification. [Online]. Available: http://opennap. sourceforge.net/napster.txt.

[12] (2012) The gnutella protocol specification v0.6. [Online]. Available: http://rfc gnutella.sourceforge.net/src/rfc-0 6-draft.html

[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. of The 2001 ACM Conference on Applications, Technologies, Architectures, and protocols for computer communications*, New York, NY, USA, August 2001, p. 161172.

[14] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peertopeer lookup service for internet applications," in *Proc. of The 2001 ACM*

Conference on Applications, Technologies, Architectures, and protocols for computer communications, San Diego, California, USA., August 2001, p. 180200.

[15] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proc. of The IFIP/ACM nternational Conference on Distributed Systems Platforms Heidelberg*, London, UK, August 2001, p. 329350.

[16] B. Y. Zhao, J. Kubiatowicz, and l. Antony D. Joseph, "Tapestry: An infrastructure for fault-tolerant widearea location and routing," in *Proc. of The IFIP/ACM nternational Conference on Distributed Systems Platforms Heidelberg*, Berkeley, CA, USA, April 2001.

[17] A. Oram, *Peer to Peer: Harnessing the Power of Disruptive Technologies*. 101 Sebastopol, CA, USA: O'Reilly and Associate.Inc, Fisrt Eddition, 2001.

[18] G. Perera, *Design and evaluation of new search paradigms and power management for peer-to-peer file sharing*. University of South Florida,FL, USA: PhD thesis, 2007.

[19] M. Shinohara, H. Hayashi, T. Hara, and S. Nishio, "A data transmission method using multicast in mobile ad hoc networks," in *Proc. of Int'l Conf. on Mobile Data Management MDM*, Taipei, Taiwan, May 2009, pp. 193–198.

[20] M. Shen, M. Kumar, S. K. Das, and Z. Wang, "Energy-efficient caching and prefetching with data consistency in mobile distributed systems," in *in IPDPS04: 18th In-*

ternational Parallel and Distributed Processing Symposium, Shanghai, China, May 2004, pp. 67–77.

[21] *A Short Look on Power Saving Mechanisms in the Wireless LAN Standard Draft*, IEEE Std. 802.11, 1997.

[22] 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 (WMCSA99)*, Sun Microsyst. Labs., Adv. Dev. Group, Menlo Park, CA, US., February 1999.

[23] P. D. Linh, *A Study for Peer-To-Peer File-Sharing Application in Cellular Mobile Networks*. Waseda University,Tokyo, Japan: Master thesis, 2010.

[24] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," in *Proc. of (ACM) The 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, New York, NY, USA, Augest 2009, pp. 407–418.

[25] S. Schmid and R. Wattenhofer, *Structuring Unstructured Peer-to-Peer Networks*. Springer US, 2007, ch. High Performance Computing.

[26] D. Tsoumakos and N. Roussopoulos, "Analysis and comparison of p2p search methods," in *InfoScale '06 Proceedings of the 1st international conference on Scalable information systems*, New York, NY, USA, June 2006.

[27] A. Datta, "Mobigrid: Peer-to-peer overlay and mobile ad-hoc network rendezvous - a

data management perspective," in *In Proc. of the CAiSE 2003 Doctoral Symposium*, Klagenfurt, Austria, June 2003.

[28] (2012) OMNET++ network simulation. [Online]. Available: http://www.omnetpp.org

[29] (2012) The oversim p2p simulator. [Online]. Available: http://www.oversim.org/wiki

[30] (2012) INET framework. [Online]. Available: http://inet.omnetpp.org/

[31] (2012) INETMANET. [Online]. Available: http://github.com/inetmanet/inetmanet