NOTE TO USERS

This reproduction is the best copy available.

UMI®

RESOURCE-AWARE TOPOLOGY ADAPTATION IN P2P OVERLAY ADHOC NETWORK

By

Gabriel Lau

(Bachelor of Applied Science, University of Toronto, June 2001)

A thesis presented to Ryerson University

in partial fulfillment of the requirements for the degree of

Master of Applied Science

in the program of Electrical and Computer Engineering

Toronto, Ontario, Canada 2004

Abstract

Resource-Aware Topology Adaptation in P2P Overlay Adhoc Network
Gabriel Lau
Master of Applied Science
Electrical and Computer Engineering
Ryerson University
Toronto, 2004

With the emergence of wireless devices, service delivery for mobile ad hoc networks (MANET) has started to attract a lot of attention recently. 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. We propose a P2P system for MANET, RAON, which performs query forwarding and overlay topology adaptation based on link instability and power constraints. We evaluated and compared the performance of RAON with an existing P2P system, Gia. Our simulation results show that RAON improves the success rate and delay of query search as compared to Gia. It, however, achieves this at the expense of higher energy consumption.

Acknowledgements

I would like to thank my thesis supervisor Professor Muhammad Jaseemuddin and co-supervisor Dr. Govindan Ravindran, for providing me direction and guidance during this work. Special thanks to Professor Jaseemuddin, who gave me countless precious advices and helped me clarify my ideas.

I also want to thank Dr. Yatin Chawathe of Intel Research and Dr. Scott Shenker of UC Berkley for providing me the simulation code of Gia.

Finally, I want to express my gratitude to my family, who constantly supported and encouraged me throughout the course of this thesis.

Table of Contents

DECLARATION ..... **II**

ABSTRACT ..... **III**

LIST OF FIGURES ..... **VIII**

LIST OF TABLES ..... **XI**

LIST OF ABBREVIATIONS ..... **XII**

CHAPTER 1 INTRODUCTION **1**

1.1. Motivation 2

1.2. Objective 4

1.3. Thesis Outline 5

CHAPTER 2 BACKGROUND **6**

2.1. MANET 6

2.1.1. MANET Unicast Routing Protocols *6*

2.1.2. AODV *9*

2.2. Peer-to-Peer Overlay Networks 11

2.2.1. Introduction to P2P file sharing systems *12*

2.2.2. Unstructured P2P System – Gnutella *13*

2.2.3. Structured P2P System – CAN *15*

2.2.4. Network Traffic of P2P File Sharing Systems *17*

2.2.5. Comparison of Unstructured and Structured P2P Systems *19*

2.2.6. P2P for Multicast *20*

2.2.7. P2P for Telephony Services (VoIP) *22*

2.3. Power Management Algorithms 24

2.3.1. Suspend/Resume *26*

2.3.2. Data Reduction *27*

2.4. Summary 28

CHAPTER 3 DESIGN 29

3.1. A Scalable Gnutella-like P2P System – Gia 29

3.1.1. The reasons for Gia *29*

3.1.2. Gia Design *30*

3.2. Resource-Aware Overlay Network 34

3.2.1. Gia's Adaptability to MANET *35*

3.2.2. Neighbor Coloring Scheme *38*

3.2.3. Proactive Neighbor Replacement *42*

3.2.4. Energy-Aware Topology Adaptation and Flow Control *47*

3.3. Summary 49

CHAPTER 4 IMPLEMENTATION DETAILS 50

4.1. Packet-level Simulation 50

4.2. Prototype Overview 51

4.2.1. RAON Application *52*

4.2.2. RAON Agent *52*

4.3. Forwarding Engine 53

4.4. Disconnect Protocol 54

4.5. PNR Implementation 56

4.6. Summary 60

CHAPTER 5 EVALUATION 61

5.1. Simulation Setup 61
5.2. P2P Query Forwarding Performance 64
5.3. Network-level Analysis 73
5.4. Energy Consumption 79
5.5. PNR Overhead 82
5.6. MANET P2P Open Questions 83
5.6.1. Reverse path for responses 83
5.6.2. TCP or UDP 84
5.6.3. Effectiveness of Overlay Flow Control 85
5.7. Summary 86
REFERENCES 91
APPENDIX A 96

List of Figures

Figure 1.1: Example of single-hop and multi-hop wireless networks 2

Figure 1.2: An example of overlay on top of MANET. 3

Figure 2.1: Route Discovery process in AODV 10

Figure 2.2: Searching for a file in a P2P network 13

Figure 2.3: Example of 2-dimensional CAN 17

Figure 2.4: Power state machine. 24

Figure 3.1: An example of query search in Gia. 37

Figure 3.2: The resulting topology of Figure 1.2. 41

Figure 4.1: Overview of the RAON Prototype 51

Figure 4.2: The operations involved in a query search. 53

Figure 4.3: Disconnect protocol state diagram 56

Figure 4.4: PNR flow diagram 57

Figure 5.1: Percent difference in colored links used by Gia and RAON for the 20:1000 scenario 66

Figure 5.2: Query Success Rate for Gia, (RAON – PNR), and RAON under different MANET configurations 68

Figure 5.3: Query delay for Gia, (RAON – PNR) and RAON under different MANET configurations 68

Figure 5.4: Average query success rate and query delay by keeping the simulation area constant in (a) and (b), while keeping the node speed constant in (c) and (d). 69

Figure 5.5: Histogram (bar) and CDF (line) of query delay distribution 72

Figure 5.6: Query Success Rate for Gia, (RAON – PNR), and RAON under different system configurations and network conditions after excluding unacceptably high delay entries 73

Figure 5.7: Query delay for Gia, (RAON – PNR) and RAON under different system configurations and network conditions after excluding unacceptably high delay entries 73

Figure 5.8: AODV Route Failures for Gia, (RAON – PNR) and RAON under different system configurations and network conditions 75

Figure 5.9: Link failure example. The dotted lines represent virtual links, and LF1 and LF2 are link failures 1 and 2 respectively. 75

Figure 5.10: AODV Requests for Gia, (RAON – PNR) and RAON under different system configurations and network conditions 76

Figure 5.11: Average physical hop counts between the neighbors for Gia, (RAON – PNR) and RAON under different system configurations and network conditions 78

Figure 5.12: Standard deviation of physical hop counts between the neighbors for Gia, (RAON – PNR) and RAON under different system configurations and network conditions 78

Figure 5.13: Maximum physical hop counts between the neighbors for Gia, (RAON – PNR) and RAON under different system configurations and network conditions 78

Figure 5.14: Number of dead nodes for Gia, (RAON – PNR) and RAON under different system configurations and network conditions 80

Figure 5.15: Dead time for Gia, (RAON – PNR) and RAON under different system configurations and network conditions 80

Figure 5.16: Normalized dead time for Gia, (RAON – PNR) and RAON under different system configurations and network conditions 80

Figure 5.17: Energy consumption pattern for all RAON nodes in the 20:1000 scenario with 50 peers 81

Figure 5.18: Isolating the energy consumption of one of the nodes in Figure 5.12 82

List of Tables

Table 3.1: Conditions for changing a neighbor's color. The symbols && and || refers to the logical AND and OR operations respectively. 40

Table 3.2: Node discovery methods. In each of the methods, it is assumed that node n is the source. 44

Table 5.1: Network level simulation parameters 62

Table 5.2: P2P level simulation parameters 64

Table 5.3: Average number of messages generated over all MANET scenarios for Gia, (RAON – PNR), and PNR. 77

List of Abbreviations

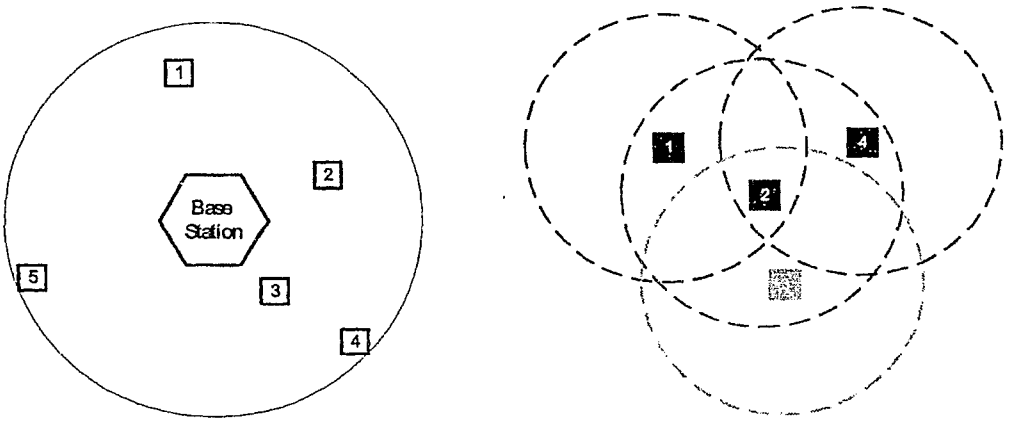
ACK	Acknowledgment
ACPI	Advanced Configuration and Power Interface
AODV	Ad-hoc On-demand Distance Vector routing
API	Application Programming Interface
APM	Advanced Power Management
AS	Autonomous System
BIOS	Basic Input Output System
CAN	Content Addressable Network
CDF	Cumulative Distribution Function
CPU	Central Processing Unit
DHT	Distributed Hash Table
DOS	Denial Of Service
DSDV	Destination-Sequence Distance-Vector routing
DSR	Dynamic Source Routing
GPRS	General Packet Radio Service
GPS	Global Positioning System
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
MAC	Medium Access Control
MANET	Mobile Ad-hoc Network
NAT	Network Address Translation
NCS	Neighbor Coloring Scheme

OS	Operating System
P2P	Peer-to-Peer
PC	Personal Computer
PDA	Personal Digital Assistant
PHY	Physical
PMCP	Power Management Control Protocol
PNR	Proactive Neighbor Replacement
RAON	Resource-Aware Overlay Network
RREQ	Route Request
RREP	Route Reply
RIP	Routing Information Protocol
RTT	Round Trip Time
TCP	Transmission Control Protocol
TTL	Time To Live
UDP	User Datagram Protocol
USB	Universal Serial Bus
VoIP	Voice over IP

Chapter 1 Introduction

Wireless devices such as cellular phones, PDA's, and laptops have become more and more important in everyone's life. These devices help people to communicate, organize, and work at anytime and anywhere. In fact, cellular phones are no longer used for voice communications only. Surveys have shown that cell-phone users are also interested in other functionalities like checking emails, using location services, downloading ring tones, taking and sending pictures, and playing games. On the other hand, PDA users also want their handheld devices to support many extra features, such as playing music, viewing albums, web browsing, and even word processing. These devices simply need to get more and more powerful in order to stay in the market. One key element that is highly demanded is to allow users to network and share contents with their peers.

In today's cellular and wireless network, mobile hosts communicate directly with a base station, and data are forwarded to the destination via the base station. This is referred as the single hop model (See Figure 1.1(a)). In contrast, a mobile ad hoc network (MANET) uses the multi-hop model where there is no central access point. A MANET is basically a set of mobile hosts forming a private network that is self-configurable. Due to limitation on radio propagation, these mobile hosts might need to act as routers and forward data to the destination for other hosts in a multi-hop fashion (See Figure 1.1(b)). This type of network is self-configurable and is viable for situations where minimum or no infrastructure support is available (e.g. disaster recovery operation, soldier communications in a battlefront, inter-vehicle notifications of accidents or traffic jams, etc.).



(a) Example of a single-hop network. The nodes communicate with each other via the base station.

(b) Example of a multi-hop network. Node 2 must act as router in order for the other nodes to communicate.

Figure 1.1: Example of single-hop and multi-hop wireless networks

Mobile hosts have complete freedom in mobility, resulting in the network topology being highly dynamic and unpredictable. Point-to-point communications rely on multi-hop routing, but if one of the links along the path failed, then the connection would be broken. For instance, in Figure 1.1(b), node 3 can communicate with node 4 via node 2. If node 2 moves out of node 3's transmission range, or node 4 moves out of node 2's transmission range, the connection would be broken. Also, wireless links generally have lower capacity than wired links, and therefore, they may suffer from congestion more frequently.

1.1. Motivation

Service Delivery on MANET has been a relatively under-explored area. It faces many challenges posed by link instability, node transience, intermittent disconnection, low battery power, and resource constrained nodes. We consider application-level overlay network as a viable service delivery architecture for MANET. An overlay

network is a set of nodes connected together forming a virtual network at the application level, independent of the underlying network. Each connection between overlay nodes is referred as a virtual link, and each link may consist of multiple physical hops. This approach is especially attractive because only overlay nodes maintain service related states. Hence, service delivery remains immune to the transience and instability caused by other nodes. Figure 1.2 shows that overlay nodes are only aware of the overlay topology, and any change in the underlying network topology is transparent to them. In fact, overlay network has been an alternative solution for introducing new services that are too difficult to deploy at the IP level (e.g. Multicast) [1].

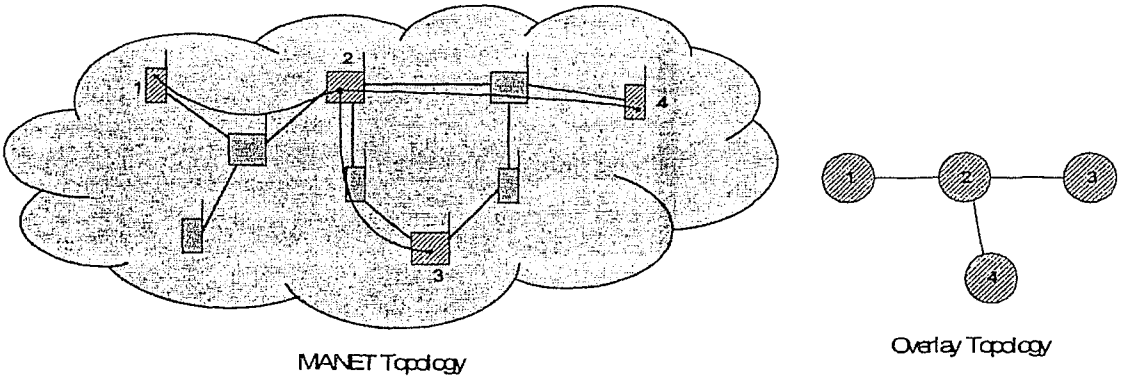


Figure 1.2: An example of overlay on top of MANET. Overlay nodes do not need to be aware of the intermediate nodes in the MANET topology.

Peer-to-Peer (P2P) is an overlay network architecture where each peer only maintains information of its neighboring overlay nodes. These peers share their resources and contents, and forward data on behalf of other peers. Every peer can act as a server and provide services to other peers, thus there is no single point of failure. Hosts may join and leave the P2P network anytime they want, resulting in rapid change of the overlay network topology. P2P systems have been widely used for file sharing

applications recently.Due to the similarities of dynamic topology and lack of infrastructure between MANET and P2P, it is attractive to design an architecture upon which P2P can adapt in the MANET environment. However, existing P2P systems are designed for wired static hosts, thus problems such as link instability and constrained power were not taken into account. The major challenges would be to locate the desired content efficiently and deliver the content to the destination reliably under the MANET environment.

1.2. Objective

The purpose of the project is to design a content delivery architecture that fits well in the MANET environment. Due to the constrained resources of mobile hosts, hot spots must be avoided in order to prevent congestion or node failure. Thus, the P2P overlay architecture is chosen to serve this purpose. However, existing P2P systems are designed for wired static hosts, thus problems such as link instability and constrained power were not addressed. Also, some contents could be more popular than the others (e.g. updated traffic and weather report), and the system should be aware of the availability of these content to the users.

This thesis focuses on the issues of P2P adaptation to the dynamic environment of MANET. Our design concentrates on providing content search service to the users. We also propose content replication mechanisms to improve content delivery efficiency and content availability through exposing node resource constraints to application-level overlay network. The objective of these mechanisms is to ensure that popular content is always available to the users through proactive replication. However, due to the design

space of this problem, they are not evaluated in this project. The following are the objectives of the thesis:

1. Research on existing P2P systems and their adaptability to MANET
2. Design an overlay P2P architecture where peers are aware of any change on their neighbors' resources
3. Proactively change the overlay topology to adapt to the dynamic underlying network topology
4. Perform packet-level simulation of the design

1.3. Thesis Outline

The rest of this thesis is organized as follows. In Chapter 2, we provide some background information on MANET routing protocols, P2P systems and their applications, and energy conservation algorithms. In Chapter 3, we address the design issues of overlay P2P for MANET. Gia, an existing P2P system, as well as our proposed system, RAON are described in this chapter. We then describe our the implementation methodology of RAON in Chapter 4. Chapter 5 demonstrates simulation results and evaluates the performance of both Gia and RAON over MANET. This chapter also contains analysis of RAON over MANET. Finally, we conclude in Chapter 6 with a summary of the thesis and with 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 MANET and MANET routing protocols. Section 2.2 describes some of the existing P2P architectures used for file sharing applications, and finally, Section 2.3 explains some proposed power conservation techniques.

2.1. MANET

In the existing Internet, the router topology is usually static, despite the rare occasion of network reconfiguration and router failures. On the contrary, MANET hosts, which also serve as routers, are mobile, leading to highly dynamic network topology. Wireless links are comparatively unstable than their wired counterparts due to the fading, noise, and interference, resulting in higher congestion rate. Also, mobile hosts are constrained in power as they typically rely on battery power [2]. All these characteristics have made routing in MANET a very challenging task to achieve. The following subsections discuss some of the existing unicast routing approaches, with a detailed description of the AODV (Ad-hoc On-demand Distance Vector) routing protocol, as it is the protocol chosen for this project.

2.1.1. MANET Unicast Routing Protocols

Among all the proposed routing protocols for MANET, they are generally categorized into 2 classes: proactive and reactive [3]. Proactive (also known as.

table-driven) routing protocols attempt to maintain a routing table with the most up-to-date routes between any pair of nodes. Routing-update messages are propagated periodically across the whole network such that each node can get a complete view of the network topology. This method enables mobile nodes to obtain routing information before any data transmission takes place. However, it sometimes suffers from wasting the limited wireless bandwidth, since every node maintains routes to any destination even if that route is never being used.

DSDV (Destination-Sequenced Distance Vector) [4] is a protocol that adopted the proactive routing approach. It uses the classical Distributed Bellman-Ford (DBF) [5] algorithm to construct the next-hop routing tables. Each node periodically advertises its own routing table to all of its current neighbors in order to maintain the most up-to-date information on the network topology. It also attaches a sequence number in every route table entry such that nodes can distinguish new entries from the out-of-date ones. This method can solve the well-known count-to-infinity problem that occurs in RIP [6].

On the other hand, reactive (a.k.a. on-demand) routing protocols create routes only when needed. A source node initiates a route discovery process for a specific destination. Typically, it broadcasts a Route Request message to its neighbors, and once the destination is found, it would receive one or more Route Reply messages. This route entry is then maintained until the source no longer needs it or no route to the destination exists. Therefore, the source does not need to waste bandwidth and energy to discover and maintain a route that it does not need at all. The downside of this approach is that when the source initiates a send request, there is a route setup time that it needs to wait before it can actually transmit the data.

DSR (Dynamic Source Routing) [7] is an example of the reactive routing approach. When a node wants to discover a route to a destination, it broadcasts a Route Request message to its current neighbors. Each Route Request is tagged with a unique Request ID. This helps intermediate nodes to identify any duplicate requests and discard them accordingly. On receipt of a non-duplicate Route Request message, if the node does not have a route to the destination, it appends its own address to the message and rebroadcasts it to its neighbors. This process continues until the message reached the destination node or a node that has a route to the destination. That node would then send back a Route Reply to the source with the complete list of intermediate nodes. The reply is propagated back to the source via the reverse path, thus all the intermediate nodes can record the complete hop sequence to the destination as well. And since the hop sequence is known, the source can eliminate any loops in the route. When it begins transmitting data to the destination, it includes the hop sequence in the packet header, and intermediate nodes just need to forward the packet to its neighbor according to the hop sequence.

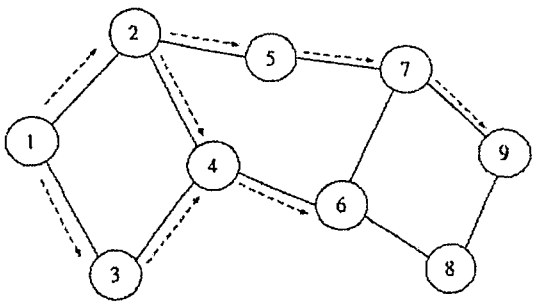
DSR does not use any periodic update messages as in DSDV. A route is maintained in the route cache and being used as long as all the links in the hop sequence still exists. If one of the links were detected to be broken, the node that "owns" the link would send a Route Error message back to the source. All the intermediate nodes along the reverse path, as well as the source node, can remove any route that uses the broken link. The source node would start a new route discovery process if there were still data to be sent to the destination. Otherwise, it does not need to respond to the Route Error message.

2.1.2. AODV

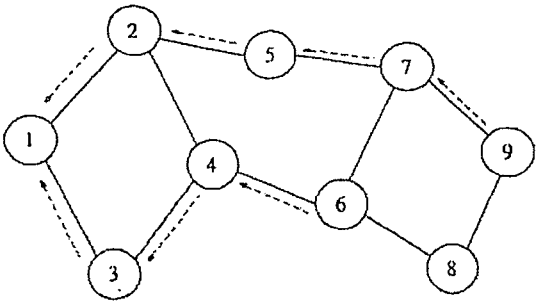
AODV (Ad-hoc On-demand Distance Vector) [8] is a reactive routing protocol that combines ideas from DSDV and DSR. Similar to DSDV, each AODV node maintains a routing table and a sequence number. Each route entry in the routing table stores the hop count, next hop, and the greatest sequence number of the destination. There is also an expiry time to indicate the validity of each route. The route entry is removed from the routing table if it is not used or updated within that time.

On route discovery, the source broadcasts a Route Request message (RREQ). Each RREQ has a broadcast ID, which is essentially same as the Request ID in DSR. Other nodes can distinguish duplicate RREQs by examining the broadcast ID and the source address. The RREQ also includes sequence number of the source itself and the destination sequence number that is currently known by the source node. When a node receives a new RREQ, it updates the source sequence number and the next hop to the source node in the routing table. If the receiving node does not have a route to the destination, it rebroadcasts the RREQ to its neighbors. If it is in fact the destination or it contains a valid route to destination with a sequence number that is greater or equal to the one specified in the RREQ, it would respond with a Route Reply message (RREP). The RREP is propagated back to the source via the reverse route, and intermediate nodes would update the destination's information in their routing tables. Thus, the forwarding route is established and maintained by each intermediate node. If a node receives more than one RREP for a specific RREQ, the destination sequence number in the RREP is used for determining which route to accept. The node would either take the one with a

higher sequence number or the one with smaller hop count if the sequence numbers were the same, or else it would simply discard the RREP. Figure 2.1 displays an example of the route discovery process, where the source (node 1) broadcasts an RREQ message and received multiple RREP messages.



(a)Node 1 initiates a RREQ for node 9. Node 6 has a valid route, so no need to rebroadcast the RREQ to nodes 7 and 8.



(b)Both nodes 6 and 9 send an RREP back to node 1. Node 1 records the one received from node 9 because it is fresher than the one from node 6.

Figure 2.1: Route Discovery process in AODV

In case of a link breakage on a route, the node that detected it can notify all the upstream nodes by sending an RREP message along the reverse path. This RREP message should contain a higher sequence number than the one currently known, and a hop count of infinity to the destination. When the source node receives this RREP message, it can reinitiate a new route discovery process if it still needs to send data to the destination. A newer version of AODV supports the Local Repair feature, i.e. if the intermediate node that detected the link breakage is closer to the destination than the source, it would initiate a RREQ and try to resolve a new route locally. Any data packet that it receives in the meantime is queued at the node, and if a new route to the destination is discovered, it flushes all the queued packets. If a new route cannot be found, it sends an RREP message along the reverse path as described above, and all the

data queued at this node are dropped.

The main difference between AODV and DSR is how the next hop routing information is determined. In AODV, this information is stored at the associated nodes on the forwarding route, and thus, routing decisions are made independently at each node. On the other hand, DSR inserts the complete hop sequence in the header of all data packets. Routing decisions are solely made by the source. So if an intermediate node found a better and more up-to-date route to the destination than the one known to the source, AODV would be able to take advantage of that while DSR cannot. And since reactive routing protocols can conserve more bandwidth and energy than proactive ones, we decided to use AODV as the routing protocol for our project.

2.2. Peer-to-Peer Overlay Networks

Peer-to-Peer (P2P) is an overlay network architecture where each peer only maintains information of its neighboring overlay nodes. These peers share their resources and contents, and forward data on behalf of other peers. Hosts may join or leave the P2P network anytime they want, resulting in rapid change of the overlay network topology. This architecture can be used to run many kinds of networking applications in a distributed fashion. Among all the applications, file sharing is the one being used the most. In this section, we describe some of architectures designed for this purpose. We also give an overview on some other applications that are deployed using P2P, namely multicast and VoIP.

2.2.1. Introduction to P2P file sharing systems

In recent years, P2P file sharing systems have been one of the most popular research topics. By definition, P2P is a set of computers that join together to form an overlay network, and file sharing is the most popular application of P2P. Napster [9] is the very first P2P file sharing system. Although it is a file sharing system, it is not truly P2P, since it uses a central server to store all the file indexes, where file index is a file pointer that indicates where the file is located.

Even Napster is not a true P2P system, it started the P2P file sharing revolution, which catches many researchers' attention to start developing more advanced P2P architectures. Typically, P2P file sharing applications allow users to share their files with other end-users, which are referred as peers. Peers join together to form a network, and they can find the desired files and download them directly from other peers. Figure 2.2 shows a simple example of searching for file in a P2P network. Each peer is a server as well as a client, and there is no single point of failure. Nodes that have downloaded the content become a content server themselves, thus servers are distributed all over the network. This is in contrast to the traditional client-server model where one server serves all clients. Hence, peers are sometimes referred as servents (SERVer + cliENT). Different P2P systems are usually distinguished by how the P2P network is constructed and how to search for the desired content within it. The actual file transmission generally happens via a direct connection from the source to the destination.

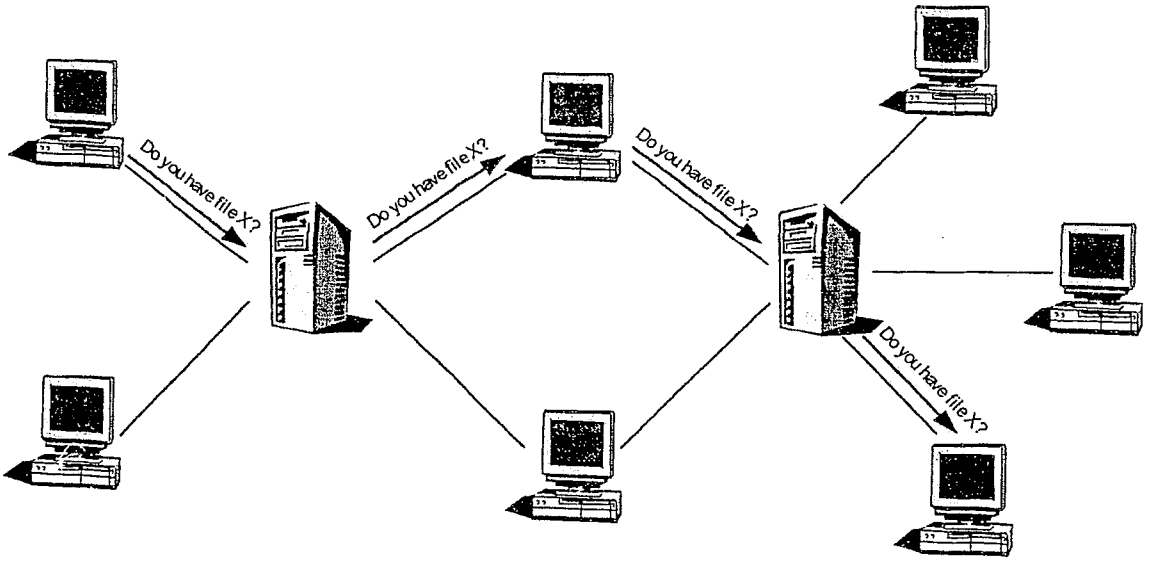


Figure 2.2: Searching for a file in a P2P network

Among the existing P2P systems, they can be classified into two categories: unstructured and structured. Unstructured P2P systems, such as Gnutella [10], have no specific way of finding the desired content, they usually do by network flooding to search the entire network. On the other hand, structured P2P systems often use distributed hash tables to directly locate the peers that contain the desired content. Thus, structured P2P systems can conserve a lot of bandwidth when compared to unstructured P2P systems. Some examples of structured P2P systems are CAN [11], Chord [12], Pastry [13], and Tapestry [14]. In the next two subsections, we study both P2P approaches in detail.

2.2.2. Unstructured P2P System – Gnutella

Gnutella [10] is one of the real P2P networks. Users join the network and setup connections to some nodes as peers at random. Since the peers are randomly selected, they form an unstructured overlay network. In order to locate a file, Gnutella floods the

network by broadcasting a query message that contains the filename or a keyword. Obviously, if a query message travels too far away, it might overload the network. Thus, a TTL value is set in the query message to limit the number of hops that message can travel. Each query message also has a universal unique identifier. When a node receives a query message, it checks the message ID. If the node has received this message before, it will drop the message. Otherwise, it continues broadcasting the message to its neighbors.

If a node contains a file that matches the query, it sends a response message back to the message originator along the reverse path, and keeps on forwarding the query to its peers if the TTL value allows it to do so. The user then selects the file that he or she wants and establishes a direct connection between the source and target node. One disadvantage of Gnutella and other similar systems is that the TTL effectively segments the Gnutella network into subnets, imposing on each user a virtual "horizon" beyond which their message cannot reach [15]. However, if the TTL were removed, the messages would be replicated in an exponential fashion throughout the network. This suggests that the Gnutella network is faced with a scalability problem [12].

The more recent version of Gnutella uses the concept of SuperNodes or SuperPeers to address the scalability issue. This concept was first proposed in the FastTrack architecture [15], which is better known by the popular client names KaZaA and Morpheus. The idea is that nodes are dynamically assigned the task of servicing a small subpart of the peer network by indexing and caching files contained in the part of the network they are assigned to [16]. Nodes with sufficient resources (bandwidth, memory, and CPU power) are automatically elected as SuperPeers. When a node wants to locate

a file, it simply sends a query to its SuperPeer, and if the SuperPeer finds the file index locally, it sends a reply back to the node. Otherwise, it forwards or broadcasts the query to other SuperPeers. Thus, the SuperPeer acts like a local server. This architecture is sometimes referred as partially decentralized unstructured P2P system, while the original Gnutella is classified as purely decentralized unstructured P2P system.

2.2.3. Structured P2P System – CAN

The idea of Content-Addressable Networks (CAN) [11] is to construct a virtual network that forms a *d*-dimensional Cartesian coordinate space. Each member, or node, in a CAN is assigned a pair of coordinates (x, y) and it "owns" a zone that (x, y) lies, given that the CAN is 2-dimensional (in the case of 3-dimensional, then each member would get (x, y, z) coordinates instead of (x, y)). This virtual space is used for storing (key, value) pairs. Using a uniform hash function, a key K_1 can be deterministically mapped to a point P within the CAN coordinate space. The key/value pair (K_1, V_1) is stored at the node that owns the zone in which P falls into. In order to retrieve this entry, a node can use the same hash function and map K_1 to point P. If the key K represents a filename of any file, and value V is the host that holds the file, then CAN is essentially a distributed, internet-scale hash table that maps filenames to their location in the network [14]. In general, most structured P2P systems use the distributed hash table (DHT) approach. Thus, structured P2P systems are sometimes referred as DHT-based P2P systems.

When a new node wants to join the network, a new node must be added to the CAN. Assuming that some external CAN bootstrap nodes exist, a new node can then retrieve

the IP addresses of the nodes in the existing CAN. The new node randomly chooses a (x, y) coordinates for itself and then sends a JOIN request to the node that owns (x, y). This node will split its zone and assign one half of it to the new node. If a node leaves the network, one of its neighbors merges with the departing node's zone, given that the merged zone is a valid single zone. If the merged zone is not valid, then one node will temporarily handle 2 zones. Each node stores a list of (key, value) pairs. So when a node joins and occupies a new zone, the node that previously owned that zone has to split its list and sends the partial list to the new node. Similarly, when a node leaves, the neighbor that is taking over the leftover zone has to get the list from the departing node.

To detect for node failures, each node is required to send periodic update messages to all of it neighbors. This update message includes the node's zone coordinates, as well as the coordinates of its neighbors. If a node dies, all of its neighbors would stop receiving the update messages from this node. All these neighbors will then start a takeover mechanism and the neighbor with the smallest zone will merge with the leftover zone. On a file search, the filename is mapped to a point in CAN, and a query message is sent to the node that owns that point. The message can be routed to the destination coordinates simply by forwarding it to a neighbor that is closer to the destination than itself. That means, if the source is at (0,0), and it wants to send data to a node at (0,3), it simply forwards it along the y-axis in the positive direction. Figure 2.3 shows an example of the 2 dimensional CAN. Each box represents a zone, and each number is the node that occupies it.

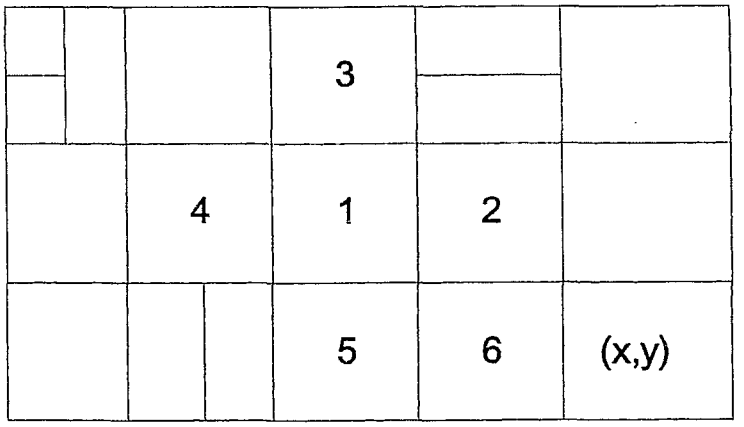


Figure 2.3: Example of 2-dimensional CAN

Some design improvements were proposed in [11] to help achieve better performance. For example, by using a multi-dimensioned coordinate space, it increases the size of the neighbor list of each node, but it reduces the routing path length of each message. Also, since each node has more neighbors, the routing fault tolerance is improved due to more potential next hop nodes. Another improvement is to increase the number of realities, where a reality is defined to be a coordinate space. This results in each node occupying a different zone in each coordinate space. Thus, the contents of the hash table are replicated on every reality, which improves data availability. Another way to improve data availability is to use multiple hash functions. Other improvements include overloading coordinate zones, using better routing metrics, topologically sensitive network construction, more uniform partitioning, and use of caching and replication techniques.

2.2.4. Network Traffic of P2P File Sharing Systems

In order to understand the P2P network traffic, three popular unstructured P2P

systems, Gnutella [10], FastTrack [17], and DirectConnect [18], were studied and analyzed in [19] (DirectConnect is another P2P system similar to FastTrack). The study in [19] was done by collecting 800 millions of flow-level records in a 3-month period. Some interesting behaviors were observed.

First of all, the network traffic volume is mostly generated by small amount of users. It is observed that the top 1-2% of the IP addresses account for more than 50%, and the top 10% of the IPs account for more than 90%, of the total traffic [19]. This suggests that the P2P search protocol should first find one of the popular hosts, such that the search for popular objects will be found quickly and efficiently. For other uncommon objects, the search might have to go through a large number of hosts in order to locate it. It is also discovered that about 90% of the IP addresses have 10 or less connections with other users, while only 1% of all the IP addresses have connections with 80 other IP addresses. This implies that by routing packets via the heavy-hitter nodes can reduce the number of hops. On the other hand, it also implies that the network is highly vulnerable to failures of these nodes.

Another major observation is that most users do not stay in the network for long. The *on-time* of a host is defined to be the sum of all the connection durations over a period of time, which can be used to characterize how long a host stays in the network. It is observed that 60% of the IP addresses, 40% of network prefixes, and 30% of the ASes (Autonomous Systems) stay for a total of 10 minutes or less per day [19]. This suggests that the P2P network topology is highly dynamic, since users join and leave very frequently. On the other hand, this also suggests that at the prefix and AS aggregation levels, it is more stable and persistent. Thus, it might be useful to provide some

indexing or caching nodes locally, such that when a node joins or leaves, this information would not be propagated to the whole network.

Both of these observations are very important that P2P architects should consider them when designing P2P systems. Due to the high level of system dynamics, it becomes very challenging to make a large-scale structured P2P system practical. On the other hand, it is suggested that by inserting indexing or caching nodes to the network, it may help to reduce the effect of dynamism in the system [19]. Gia [20] is a system similar to Gnutella, but take into account the two observations mentioned above, which makes it more practical and scalable. The details of the Gia design will be described in Section 3.1, as it is the core of this thesis.

2.2.5. Comparison of Unstructured and Structured P2P Systems

Both structured and unstructured P2P systems have advantages and disadvantages. In a purely unstructured system such as Gnutella, essentially nothing needs to be done for maintaining the overlay network. Due to the behavior of P2P users, this in fact is a better way to adapt to the dynamism of P2P systems. However, it creates problem when one needs to locate some particular data, which it can only rely on flooding the network. On the other hand, structured systems provide very efficient and robust query forwarding techniques. By applying a predefined hash function to the desired content, the requesting node can essentially determine which node can provide a match for its query before actually sending out the message. Therefore, no real search is required, as a query is forwarded to a specific destination, and the destination node would provide a reply for the source node. This is accomplished by organizing the P2P nodes to form a

network in a virtual space where they follow a set of common rules. However, this approach is not easy to support keyword search, which is a very important feature in P2P file sharing applications. Furthermore, it is not as adaptive as unstructured systems when users join and leave. These problems will be revisited and discussed with more details in Chapter 3.

It is interesting to note that neither structured nor unstructured systems consider routing efficiency as a major factor when constructing the overlay. This is due to the fact that P2P file sharing systems only use the overlay to direct the queries rather than content, where queries are comparatively much smaller in size. Content is transferred through a separate connection outside the overlay network. Thus, the routing efficiency problem is not a big factor in P2P file sharing systems. However, this factor can be significant for applications like multicast and telephony systems, as described in the following subsections. Routing efficiency and robustness can greatly affect the performance of these applications since data is transferred over the P2P links.

2.2.6. P2P for Multicast

Due to the lack of reliability, congestion control, flow control, and security of IP multicast, application-level multicast has become an alternate solution to provide such services. Since P2P is essentially an overlay network that is built at the application level, it is feasible to support multicast service on top of it. Some proposed ideas for P2P multicast are CAN-Multicast [21] and Scribe [22]. CAN-Multicast is based the CAN design described earlier in Section 2.2.3, and Scribe is based on the Pastry [13] P2P system. In short, Pastry is another structured P2P system that adopts the distributed

hash table approach. It uses a circular 128-bit namespace, as compared to the coordinate space used in CAN, and each node has a *nodeId* that is chosen randomly and uniformly. Similar to CAN, keys are used to represent objects, and each node is responsible for all the keys that are numerically closest to its *nodeId*. Each node also maintains a list of neighbors with *nodeId*s closest to itself. Thus, routing in Pastry is simply forwarding the message to a neighbor with *nodeId* closer to the key than itself. Since structured P2P systems provide some kind of routing mechanisms, they are more suitable to provide multicast service than unstructured P2P systems.

There are mainly 2 approaches for P2P multicast: flooding-based and tree-based. Flooding-based means for each multicast group, a separate P2P overlay is created. In such cases, multicast is simply broadcasting to every node on the overlay. This approach has the advantage that only group members need to maintain multicast information as well as transmitting the data. However, constructing a new overlay per group introduces more overheads to multicast group members. The CAN-Multicast described in [21] uses the flooding technique. On the other hand, the tree-based approach builds a spanning tree for each multicast group. Scribe in [22] adopts this method and each member is connected to the root of the tree through multiple P2P connections. There is a key called *groupId* that identifies each multicast group, and the node that is responsible for this key becomes the root. When a node wants to join a group, it sends a join message to the root. Any intermediate P2P node that receives this message would add the source node into its children table for the specified group. It would also regenerate a join message for itself and forward it towards the root, regardless if it is a member or not. This way, a reverse path is established and data packets can be

sent from the root to each group member. Scribe utilizes all nodes in the system, including non-members, to help forward and duplicate packets for the group members.

Although CAN-Multicast and Scribe are proposed with different approaches, it is suggested that multicast is independent of the P2P network structure [23]. This means that the tree-based approach can be used in a CAN network, and the flooding approach can be applied in a Pastry system. In [23], the authors made a head-to-head comparison between the two multicast methods on both CAN and Pastry, with a total of four combinations. It is discovered that tree-based approach outperforms the flooding approach in terms of delay and overhead for either P2P systems. This is mainly due to the expensive cost of overlay construction for the flooding method [23].

2.2.7. P2P for Telephony Services (VoIP)

Internet-based telephony services have been around for years. Some existing telephony applications use a centralized server to maintain the directory of users and to route each and every call [24]. That means, when a user wants to call a friend, it needs to send a request to the server to lookup the IP address of that friend. Then when the call is made, the voice packets need to go through the server again, which redistributes them to the destination. With this approach, as the user base grows, the costs for these servers scale proportionally, with the quality and reliability of the service degrade at the same time. Another major problem of existing telephony applications is that clients that are hiding behind a firewall or NAT (Network Address Translation) are very difficult to reach, which makes the call-completion rates very low.

The founders of KaZaA has moved a step forward and decided to apply

voice-over-IP (VoIP) on top of the P2P network. They have deployed a P2P telephony application, Skype [24], which is built upon the FastTrack technology used in KaZaA. Recall that FastTrack uses the SuperNode concept, thus the user directory lookup problem is solved automatically, which is essentially the same as filename lookup procedure. As the number of users grows, the number of SuperNodes will grow proportionally as well. Also, since the voice packets do not need to go through a centralized server, instead they are sent directly between the callers, this approach again solves the quality and reliability bottleneck problem. Skype uses a proprietary routing algorithm to route the voice packets over the P2P network. This approach helps to solve the Firewall problem, since clients that are not behind a firewall or NAT can now help to route packets to those clients that are hidden. Since the P2P connections are established when the clients join the network, no extra work needs to be done to route through firewalls or NATs. Another advantage of the P2P approach is that it is not vulnerable to denial-of-service (DOS) attack due to the fact that it does not have any central point of failure. Furthermore, all data packets are encrypted, which is an essential feature since all calls are routed through the public Internet. Recently, Skype has added additional features such as conference calls and PC-to-phone calls in the application.

Although Skype has shown success in supporting VoIP using an unstructured P2P system, we should not ignore the potential of delivering this service with structured P2P systems. As suggested in the previous subsection, structured P2P systems supports some kind of routing mechanisms in the overlay, thus they are good for forwarding query as well as data packets. Therefore, further studies should be made to explore the possibility of implementing telephony applications using structured P2P systems.

2.3. Power Management Algorithms

Power is the most valuable resource for any electronic device. For wireless devices, this is a more critical issue since they often rely on batteries, so power is limited. There are many power management techniques being proposed and they mainly focus on non-communication components such as display, CPU, and disk. These peripherals drain a lot of power when they are active. Thus the principle behind power management is to suspend them whenever the user does not need them, and resume them when the user wants to activate them. Several power states can be employed to define the inactiveness of the machine. Figure 2.4 shows an example of such a state machine.

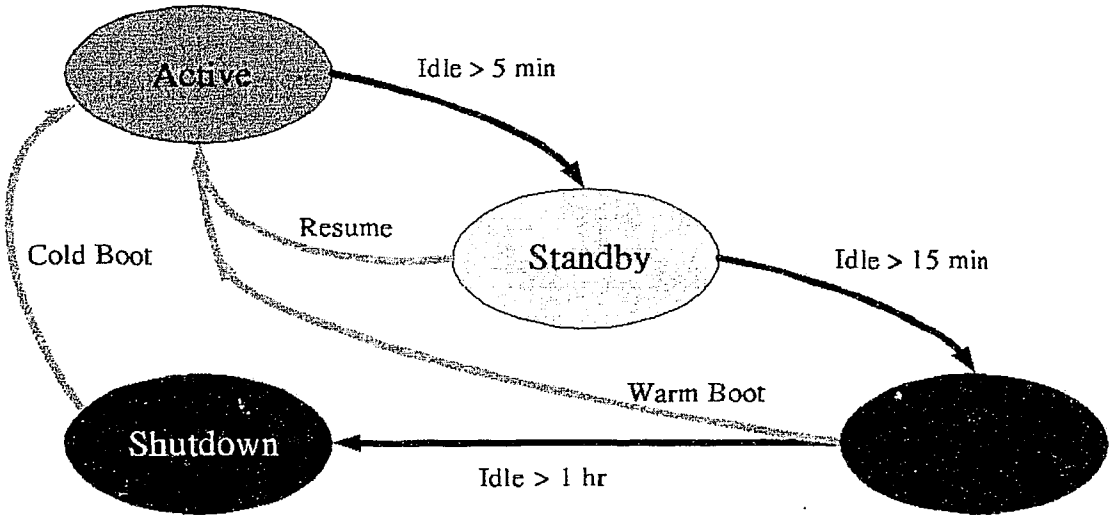


Figure 2.4: Power state machine. Different actions can be done to save power at each state, e.g. at "Standby" state, activate screensaver, spin down the hard disk, slow down the CPU; and at "Suspend" state, turn off monitor and hard disk, and stop the CPU.

APM (Advanced Power Management) and ACPI (Advanced Configuration and Power Interface) are two existing standards that applied this technique for laptop computers [25]. In APM, the power management decisions are made by the BIOS,

while this is done at the OS level in ACPI. The major drawback of APM is that the BIOS does not have any information on application-specific requirements, thus it might misinterpret the users intention suspend a component where it is not supposed. It also doesn't have knowledge about USB and IEEE 1394 components. Therefore, ACPI moved the power management module to the OS level instead.

This technique works very well for the components mentioned above, since the OS can use different power states to gradually lower the power consumption rate as the user's inactive period increase. However, for communication components such as 802.11 and bluetooth, the inactive period is not only determined by the local host, but also depends on any remote host that is trying to communicate with it. There are two main approaches that can be used for conserving power for communication. The first one is to use the same suspend/resume cycle mentioned above. Communication components need to be active in order to receive data, so the power management module is required to determine when to suspend and how long to suspend really carefully. The second approach is to reduce data size using techniques such as data compression, thus reducing transmission time. However, this method must rely on application programmers to provide this functionality. There are also other solutions that are implemented at the hardware-level, specifically at the MAC (medium access control) or PHY (physical) layer. The key disadvantage of this approach is that it has no information on application-specific behavior. In the remaining of this section, we look into examples that employ each of application-level approaches.

2.3.1. Suspend/Resume

Kravets et al. [26] presents a power management control protocol (PMCP) that achieves power management by offering functionality of suspending and resuming the communication device at the transport layer. It provides an interface for applications to define a policy that specifies how the suspend/resume routine should take place. This protocol is designed based on the one-hop wireless model, i.e. mobile hosts communicate with a base station. It further assumes that all point-to-point communications are done through the same base station.

PCMP is a protocol that operates between a mobile host and a base station, where the host is the master and the base station is the slave. When the master is suspended, it sends a SLEEP message to the slave to notify it. In this state, any data that is targeted for the master would be queued at the slave. The slave waits until it receives a WAKE_UP message from the master, then it would forward all the data that it has buffered to the master. This suspend/resume cycle results in an idle/bursty communication routine between the host and the base station. If the suspension period is too long, the bursty traffic can lead to unwanted retransmissions, which means consuming more power. It also introduces higher latency to all transmissions. If the period is too short, the power conserved might not be acceptable. Thus, there is a tradeoff between battery life and transmission delay, and the application must specify what kind of tradeoff is appropriate to it.

There are two timing-related parameters that PMCP need to determine: when to suspend and the duration of it. Both of these parameters are dependent on the application. Ideally, whenever there is no traffic on the network, the communication

devices should be suspended. Although this is impossible to accomplish, there is usually a communication pattern for each application, which could help mobile hosts to predict the idle periods. The host can listen to the medium for a period of time, and if there's no traffic within this time range, it can conclude that communication is idle and it can suspend the device. This timeout period cannot be too long or too short, thus it should be supplied by the application. After the device is suspended, it must determine how long it should stay in that state. Since the acceptable level of delay and average data size vary from one application to another, this parameter must be provided by the application as well.

Simulation results suggest that this power management control protocol can conserve up to 83% of communication power, using the communication pattern of a web user. On the down side, it poses an extra delay of 0.4-3.1 seconds. This number might be acceptable for web browsing applications, but absolutely not for real-time applications such as video conferencing and multiplayer games. However, real-time applications usually do not have too much idle time, thus PMCP would not be able to gain any benefits, instead it would create more overhead due to additional control messages.

2.3.2. Data Reduction

It is known that for any wireless device, the power consumed is proportional to the size of transmitted data. Therefore, reducing the data size is another method to cut down the power consumption rate. In [27], Flinn et al. demonstrated how data fidelity could affect the device's battery life. They investigated 4 types of applications, namely video player, speech recognizer, map viewer, and web browser. For each application,

different levels of fidelity reduction methods are applied. For example, the video player can achieve fidelity reduction by increasing the amount of lossy compression or decreasing the display window's size of the video clip. Experimental results show that lowering data fidelity does help to conserve energy, but the effectiveness of it is vastly dependent on the application type [27].

2.4. Summary

In this chapter, we reviewed some of the MANET routing protocols that are commonly used, and focused on AODV, which is the protocol selected for this thesis. We then provided an overview of several existing P2P systems used for file sharing applications and compared their differences. We also described some other applications that can be supported by P2P. At the end, we discussed some energy conservation algorithms specifically designed for reducing the energy consumed by communication components of wireless devices. In the next chapter, we discuss the design issues of the thesis, and propose a solution that addresses those issues.

Chapter 3 Design

The goal of the project is to design a P2P system that is suitable for deployment on MANET. Due to the constraints and dynamism of MANET nodes, the P2P system must be aware of the resources available and make adaptations to any change. Among the existing P2P systems, Gia is the one that provides the features that MANET can take advantage of. We decided to use Gia as the basis, and extend it to address the problems that exist in MANET. We have also chosen to use reactive routing, specifically AODV, as the underlying routing protocol. This chapter explains the motivation of Gia and the approach that it took. It is accompanied by a detailed description of our MANET P2P system, Resource-Aware Overlay Network.

3.1. A Scalable Gnutella-like P2P System – Gia

Gia is an unstructured P2P system that combines numerous schemes that were proposed for improving the scalability and robustness of such system. This section discusses the problems of unstructured P2P networks that Gia addresses, followed by the design overview of Gia.

3.1.1. The reasons for Gia

As described in section 2.2.3, structured P2P systems use the concept of distributed hash tables, where all the file indexes are distributed evenly among the users. Thus, any file in the network can be located simply by sending a query to the node that has the file pointer, and the file pointer provides a mapping between the file and the location of it. Although this technique is much more scalable than unstructured P2P systems, it only

supports exact-match queries, which means the identifier of the requested object (e.g. filename) must be known. To allow keyword search in such systems, a typical approach is to construct an index per keyword [28]. However, this method is very costly in terms of maintenance due to frequent node join and leave.

Another disadvantage of structured systems is that it is very hard to maintain the structure required for routing in a very transient node population, in which nodes may join and leave at a high rate [29]. As discussed in section 2.2.4, P2P clients are extremely transient, thus it causes a significant overhead for the system to maintain a stable state. In case of abrupt node failure, it would take even more time and effort to recover from the failure. This is because it takes time for the system to detect the node failure (due to loss of refresh messages). After the failure is detected, the file pointers that were previously stored in the failed node need to be retrieved and transferred to another node or nodes. In contrast, both of these problems do not affect unstructured systems like Gnutella at all. Thus, a new P2P file-sharing system, Gia, that is built upon the Gnutella design is proposed in [20].

3.1.2. Gia Design

The most critical problem of Gnutella is scalability due to the flooding mechanism used in the search protocol. The concept of SuperNodes used in the FastTrack technology, as well as the more recent version of Gnutella, helps to improve Gnutella's scalability. Recently, instead of broadcasting the queries to all the peers, the idea of random walk is also proposed in [30]. Although this method greatly reduces the number of queries generated at each node, it is essentially a blind search that may increase the

search time significantly. Gia modified Gnutella to incorporate the ideas of SuperNode and random walk. The Gia design consists of four main components, i) dynamic topology adaptation, ii) active flow control scheme, iii) one-hop replication of file pointers, and iv) search protocol.

The topology adaptation algorithm is used for building the overlay topology. Each node is assigned a capacity level, which is defined to be the number of queries the node can handle per second. The number of connections a node makes to other nodes depends upon its capacity level, the higher the capacity, the more connections it'll make. Neighbors inform each other about their capacity levels and current degree when the connection between them is initially established. They also send periodic update messages to notify each other on any degree change. Each node independently computes a *level of satisfaction (S)*, which is a value between 0 and 1 that represents how satisfied a node is with its current neighbors. A value of 0 means it is not satisfied at all and it would look for new neighbors, while a value of 1 means it is fully satisfied and no need to make new connections. This value is calculated by adding the capacities of all the neighbors (normalized to their degrees) and divides the sum by the node's own capacity. This ensures that high capacity nodes have more neighbors than low capacity nodes, in which they can act as SuperNodes in some sense. This number is also used to determine the adaptation interval, which dictates the aggressiveness of the topology adaptation algorithm.

In order to establish new connections, a Gia node (say X) must identify other existing peers first. Node discovery can be achieved through contacting a well-known bootstrap server or make use of ping-pong messages as described in the Gnutella protocol [31].

After a list of existing peers is revealed, it keeps this list of peers in its cache (servent cache), and X chooses a candidate from the list (say Y) with the highest capacity and tries to connect to it. When Y receives a connection request, it would accept the connection automatically if it has not reached its maximum number of neighbors (each node has a parameter *max_nbr* that is determined by its capacity). Otherwise, it would compare X's capacity with its existing neighbors' capacities, and if by allowing X to become a new neighbor can help Y to be more satisfied, then Y would accept the request and drop one of its neighbors. More details on whether to accept a new connection or not and deciding which neighbor to drop can be found in [20].

In order to avoid overloading any node with queries, Gia included a flow control scheme. Every node has a pool of tokens, and it periodically distributes them to its neighbors proportional to their capacities. These tokens represent the amount of query messages the node is willing to accept from its neighbors. Every time a node sends a query to a neighbor, it consumes a token that it has received from that neighbor. If a node uses up all the tokens from a particular neighbor, it cannot forward queries to that neighbor until it receives a new set of tokens again. The token assignment algorithm is based on Start-time Fair Queuing (SFQ), where the neighbor's capacity is used as weight. The tokens that were assigned to inactive neighbors are automatically redistributed proportionally among the other neighbors. And as neighbors join and leave, the tokens would be redistributed accordingly.

Gia also implements one-hop replication, where each node maintains an index of content of all of its neighbors. Since the topology adaptation algorithm causes high capacity nodes tend to have more neighbors, this leads them to contain a larger file index

list as well. Given this characteristics, Gia uses a biased random walk as its search protocol, where a node forwards a query to the neighbor with the highest capacity, given that it has a valid token received from that neighbor. This protocol is based on the intuition that high capacity nodes can often provide useful responses for large number of queries. Each query has a globally unique identifier (GUID). If a node receives a query that it has served before, then it would forward the query to a different neighbor. Associated with each query are the parameters *max_responses* and *TTL*. The *TTL* value is decremented every time the query is forwarded, and *max_responses* is decremented whenever a query hit is sent. A query is dropped if either of these values reaches zero, thus they dictate how far a query can travel.

In summary, Gia is a system that takes advantage of the benefits of unstructured systems, and improves the scalability at the same time by controlling how the network is constructed. Due to the topology adaptation protocol and one-hop replication, Gia is able to use a biased-random walk to provide a higher success rate in query search. According to a research done at the University of Maryland where they compared the performance of many existing unstructured P2P search methods, and it was discovered that Gia is "a very good all-around solution, combining different ideas from other schemes" [32]. Also, the topology adaptation protocol and the flow control mechanism made the P2P topology and query forwarding to be aware of the nodes' capacities. Since resources are very limited in the MANET environment, we believe that Gia's design may fit well in it, so we decide to use Gia as the basis of our project.

3.2. Resource-Aware Overlay Network

P2P and MANET share the common characteristics of dynamic topology and lack of infrastructure, thus we consider providing services for MANET users with the P2P framework to be a feasible solution. Existing P2P systems are mainly designed for stationary hosts that have reliable network connectivity, high processing power, and virtually unlimited energy supply. Unfortunately, none of these assumptions is applicable to MANET hosts. Therefore, we need to reevaluate these systems under the new environment.

Structured P2P systems require each node to hold indexes for any node within the network, despite that the index might be pointing to a node that is far away from itself, both virtually and physically. If a node leaves the P2P network, the file indexes that it is holding needs to be maintained by another node, and this operation is rather expensive. And in case of a graceless failure, where a node crashes without any notice, the recovery operation is even more expensive since neighbor nodes must first detect the failure, and then retrieve the pointers that were stored in the failed node. In the MANET environment, network connections can be disconnected instantaneously due to node mobility and limited battery life. This poses a huge problem for structured P2P systems since graceless failures can happen frequently. Furthermore, a node (say X) may have steady connection with one neighbor (say Y), but fragile connection with another node (say Z). This can cause a P2P network like CAN to go into an unstable state, where Z thinks X has failed and attempts to takeover X's zone. Z might eventually find out from Y that X is still active, but valuable resources have already been wasted during the takeover process. In contrast, node failures do not affect unstructured overlay networks

that significantly. Therefore, we consider unstructured P2P systems to be a better solution than structured ones.

Among all the unstructured P2P architectures that we explored, we believe that Gia's design may fit well in the MANET environment, although it is not specifically designed for deploying on MANET. In this thesis we propose a P2P system for MANET, Resource-Aware Overlay Network (RAON), which essentially uses Gia as the foundation, and adds the features to address problems specific to MANET. In this section, we first discuss some of the problems that Gia might face when running on MANET, followed by a detailed description of the RAON design.

3.2.1. Gia's Adaptability to MANET

The abstraction of capacity level of a node plays a significant role in all four design features of Gia - topology adaptation, flow control, one-hop replication, and biased random walk. Topology adaptation ensures that high capacity nodes are highly connected, while low capacity nodes are not. The flow control mechanism uses tokens to limit the number of queries a node can send to another, where the number of tokens each neighbor receives is determined by its capacity. One-hop replication makes high capacities nodes to contain more information. Finally, biased random walk increases the probability of query hit by forwarding queries to high capacity nodes.

In Gia, capacity of a node is defined to be the number of queries it can handle within a period of time (typically per second). This value is mainly determined by the CPU speed, memory, and bandwidth of the node, which are usually constant for a static wired node. Thus, the capacity level of a node remains constant in Gia throughout the node

lifetime. However, this is only true for static wired nodes, where their location and bandwidth do not change significantly, and they have unlimited power. In contrast, MANET nodes have the characteristics of unpredictable mobility, link instability, and limited power. These factors could affect both the node's capacity and the overall network performance.

The biased random walk algorithm forwards queries to high capacity nodes. This method greatly reduces the number of queries caused by the flooding (broadcast) algorithm, but at the same time offers a higher probability of getting a query hit than a pure random walk. This approach is favored for the MANET environment since it greatly reduces the number of messages generated per query. However, since Gia is not specifically designed for mobile networks, it does not take into account of the constrained resources of mobile nodes. In MANET, capacity is no longer a constant value, rather it changes based on the underlying network condition.

Since the biased random walk always tends to forward queries to high capacity node, the links to those nodes would experience more traffic than other links. If one of those links were unstable, it would affect the performance of any query that passes through the high capacity nodes. The performance of a query search is defined as the time required for the query originator to receive a query hit response for a particular query. Unlike flooding where many copies of the same query are generated on each search, only one copy is generated in biased random walk. As link instability is not uncommon in MANET, the performance of a query search can suffer badly if it is not aware of the instability of the links to high capacity nodes. As shown in Figure 3.1, link CA receives more traffic than the link CB since A has a higher capacity (thus more neighbors) than B.

Likewise, link AG carries more query traffic than any other links to node A. Moreover, a high capacity node usually processes more messages than low capacity nodes, which causes it to expend power at a higher rate. Although we can safely assume that most of the wireless devices are equipped with some kind of power regeneration system (e.g. solar power), especially in the future, if the consumption rate were higher than the recharging rate, then eventually the node would run out of power and fail.

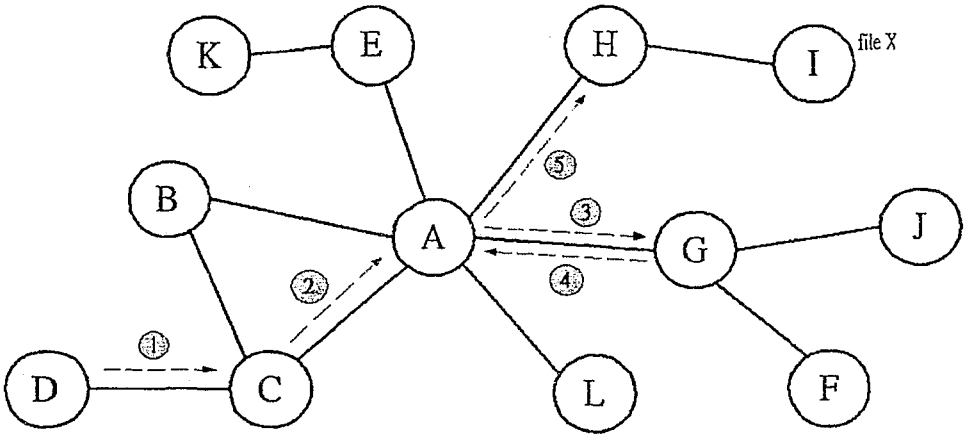


Figure 3.1: An example of query search in Gia. Peer D wants to look for file X, which is sitting at peer I. As illustrated, C would always forward the query to A first rather than B, and A would always forward to G first rather than any other neighbors, assuming that the P2P topology does not change while this search is happening.

Taking into consideration of link instability and power constraints, RAON modifies Gia's design to adapt to those constraints in MANET. We keep the notion of Gia's constant capacity level of a node, but restrict it to only reflect the node's computing power, particularly CPU speed and memory. RAON uses a ranking system to classify neighbors according to their dynamic properties, and tries to improve the overall network performance by changing the overlay topology.

3.2.2. Neighbor Coloring Scheme

The main aspect of the RAON design presented in this thesis relates to is to address the problems that Gia did not, i.e. link instability and power consumption. We propose that a RAON node keeps track of the delay it experiences with each of its neighbors and the neighbors' energy levels. Both of these values tend to change frequently over time, so we define a ranking system, called the Neighbor Coloring Scheme (NCS), for each node to categorize its neighbors into three different classes depending on their dynamic attributes. The nodes are colored GREEN, YELLOW, and RED based on their ranks ranging from High, Medium, and Low. The idea is to make the biased random walk algorithm to also take neighbors' colors into consideration when making a forwarding decision in addition to the neighbors' capacity levels. As in Gia, a RAON node requires a token in order to forward a query to a neighbor, and it only forwards to neighbors that it has not sent that query to before. Among the "available" neighbors, it groups them according to their color and selects the one with the greatest capacity from the highest color group. Thus, if neighbor A has a higher capacity but lower color than neighbor B, the query would be forwarded to B instead of A.

Algorithm 3.1 shows the steps a node needs to take in determining which neighbor to forward a query to. The addition of NCS to the biased random walk in RAON makes it avoid using unstable links (to improve query search performance) and forwarding to low-energy nodes (to increase the battery life of these neighbors). Therefore, the rank of a neighbor is lowered if either the delay is high or the energy level is low. This way, when a peer is running low in either bandwidth or power, its neighbors are aware of that and forward queries to other neighbors that have more resources instead. Therefore, the

peers are not only sharing content, they are also sharing resources among themselves.

```
Let Col_i represent the color of neighbor i
Let Cap_i represent the capacity of neighbor i
max_color ← RED
max_capacity ← 0,
max_node ← x

subset ←   i for every i in neighbors_x such that x has tokens from i
           and has not forwarded query with qid to i before
For all i in subset {
    if (Col_i > max_color)
    or ((Col_i == max_color) and (Cap_i > max_capacity))
    then {
        max_color ← Col_i
        max_capacity ← Cap_i
        max_node ← i
    }
}
return i
```

Algorithm 3.1: Node x receives a query with *qid* and determines which neighbor it should forward it to.

Each node can determine the delay by probing the neighbors periodically and measure the round trip time (rtt). The probe message can be a separate control message or piggybacked to another message. If routing information such as hop count and link layer failure detection is available to the application level, this information may benefit a RAON node where it can have a clearer view of the underlying topology, and make more accurate decisions when ranking its neighbors. However, this requires the routing agent, such as AODV, to expose this information to the application layer, which is not supported at the moment. After a node sent out a probe message to its neighbors, it sleeps for *probe_interval* seconds and waits for an ACK message in the mean time. In case that it

does not receive an ACK when the timer expires, it would consider the previous probe to be unsuccessful and assign a worst-case rtt (i.e. the *update_interval*) to that neighbor.

Energy level can be exchanged among neighbors periodically as well, but this means update messages are sent even if the energy change is very minimal, which is a waste of power. Also, the battery size usually varies from one device to another. For example, as compared to a cellular phone, a laptop has a much larger display and many other peripherals it needs to power up, so it requires a bigger battery. In such cases, the actual energy level is not meaningful to the neighbors at all, and it might be misleading at times as well. Therefore, we decided to use energy states instead of the actual value. Each node monitors its energy level and determines its energy state relative to its battery size, and updates its neighbors only when there is a transition. We define energy states of HIGH, MEDIUM, and LOW, which correspond to the three colors defined in NCS. We can also use the same technique to relate latencies to the colors by predefining the threshold values for medium and high latency. Table 3.1 summarizes NCS by showing the conditions for a node to change a neighbor's color. Furthermore, the flow control mechanism can be more adaptive to the MANET environment with the extra information that NCS provides. More specifically, flow control can assign less tokens to neighbors that are colored YELLOW, and even lesser for the ones that are RED, depending on the neighbor's energy level and/or link delay.

Neighbor's Color	Condition
GREEN	ENERGY=HIGH && LATENCY=LOW
YELLOW	(ENERGY = MEDIUM && LATENCY != HIGH) || (LATENCY = MEDIUM && ENERGY != LOW)
RED	ENERGY=LOW || LATENCY=HIGH

Table 3.1: Conditions for changing a neighbor's color. The symbols && and || refers to the logical AND and OR operations respectively.

NCS provides a scheme for the forwarding algorithm to avoid using high delay routes or consuming power of low energy neighbors. If the neighbor's color is set to RED because of low power, sending lesser queries to it can help extending its battery life and hoping that eventually it can gain access to an energy source and recharge its power. If it is due to high latency, avoiding queries to be forwarded to this neighbor might improve the performance of the search, hoping that other nodes can lead to a hit as well. High latency between two nodes can be caused by many reasons, and one of them is the randomness of the MANET topology. Connection can become unstable if the destination or any intermediate underlying node is congested, or one of the wireless links along the route is broken or experiencing interference. Recall that the MANET topology is transparent to the P2P network, thus an overlay node might remain connected to a neighbor that is many hops away from it physically. The more number of hops means a higher probability of encountering an unstable link along that route.

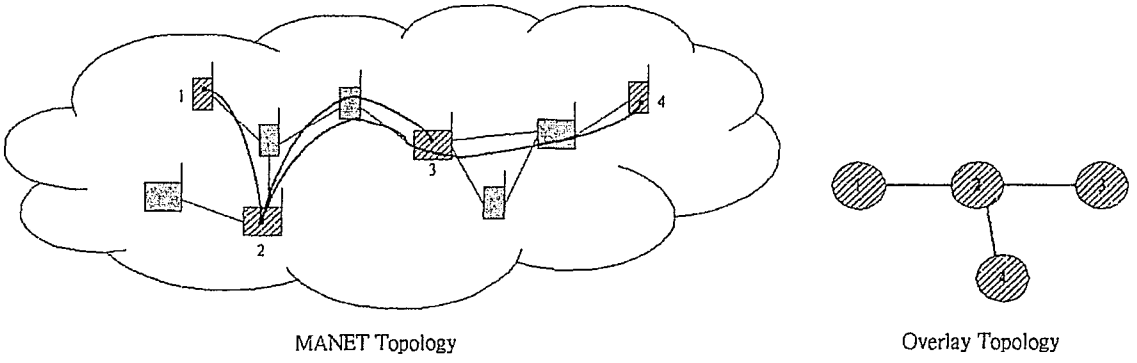


Figure 3.2: The overlay topology works well with the original topology (refer to Figure 1.2), but poorly after some node movement. The connection between node 2 and 4 now consist of 5 physical hops. There is also some overlapping routing between node 2 and 3 as well.

We can visualize the above situation in Figure 3.2, which is the resulting topology of Figure 1.2 in Chapter 1. Due to node mobility, the wireless nodes are moving freely

within the network, leading to many changes in the physical connections. However, the overlay nodes are unaware of the physical changes and remain connected to their neighbors as it was before. As illustrated in the diagram, the number of physical hops between node 2 and 4 has increased from 2 to 5. Also, this route overlaps with the route between node 2 and 3. The NCS causes node 2 to select node 1 or 3 over node 4 when forwarding queries. It tries to achieve optimal query performance by selecting the best available neighbor as the next hop without changing the overlay topology. Further optimization can be achieved by creating new link in the topology. For instance, the optimal solution in this situation is to establish a new connection between 3 and 4. Therefore, a mechanism is needed to make the overlay topology adapts to the dynamics of the underlying network. We propose Proactive Neighbor Replacement (PNR) algorithm to achieve the necessary topology adaptation.

3.2.3. Proactive Neighbor Replacement

RAON uses a Proactive Neighbor Replacement (PNR) mechanism to further improve the overall performance of NCS by changing the overlay topology adaptively. If a node n measures the latency to be HIGH with one of its neighbors, it tries to form a new neighbor relationship with an overlay node that is not currently a neighbor and is reachable through a low latency MANET route. There are two major steps involved in PNR: 1) search for a neighbor candidate; and 2) establish a new connection. Table 3.2 shows a list of node discovery methods that can be used in RAON. Each of these methods returns a list of neighbor candidates to n, and the next step is to decide which one it should connect to.

Node Discovery Method	Description
Ring Search	➢ *n* sends out a broadcast message to all the nodes within its transmission range. Any node that is a P2P node would send a reply back to it, otherwise, it broadcasts the message on behalf of *n* again until the TTL expires ➢ The message can include the existing neighbor list of *n*, such that nodes that are already neighbors would not send a reply back ➢ This method needs the underlying layers to provide APIs for the application to perform this task, since at the application level, a node cannot send a message without specifying the destination
Ping-Pong	➢ *N* sends a PING message through a link with the lowest latency. The PING message should include *n*'s existing neighbor list. ➢ Nodes that received a PING message check its own neighbor list, and if there is one that is not in *n*'s neighbor list, then send a PONG message with that node's information back to *n* via the reverse route. ➢ The PING message consists of a TTL value and it is decremented every time it is forwarded, and the message is dropped when TTL reaches zero. ➢ By forwarding the message through low latency links continuously, the probability of finding a node that *n* can establish a stable/low latency connection with is higher
GPS and GPRS	➢ With the emergence of GPRS and GPS, we can assume that the future mobile devices would mostly be equipped with these features ➢ A mobile node can communicate with a server through GPRS and update its physical location (according to GPS) periodically, the server would then have the updated location of all the nodes that are using the service ➢ When node *n* needs to discover new nodes, it can send a request to the server with its updated position. The server would then reply it with a list of nodes that are physically close to it. The request should also include *n*'s existing neighbors, such that the server could exclude those nodes in the reply ➢ Two nodes that are physically close do not necessarily mean that the connection between them is stable. The approach here is to hope that node *n* would be able to find at least one node that it can establish a stable connection, since we only need one to replace the unstable one ➢ The server can also keep track of the direction and velocity of each node, so that it can reply with a list of nodes that are traveling towards *n* (or exclude those that are traveling away from it)

Table 3.2: Node discovery methods. In each of the methods, it is assumed that node *n* is the source.

The ideal solution is to select the one with the lowest delay, but this requires *n* to probe every node in the list first. In order to probe a non-neighbor node, *n* would have to establish a TCP connection with it, and before that happens, it needs to find a route to that node. This process may consume substantial amount of bandwidth and power, which are the resources that we aim to conserve in the first place. Thus, instead of probing all the candidates, *n* chooses one with the highest capacity and probes it. If the delay were found to be in the HIGH state, *n* would send a P2P connection request to it. If the connection were accepted, *n* checks the number of neighbors it is maintaining at the moment, and if it has exceeded the upper limit, it would simply drop the unstable neighbor that it was trying to replace originally. If the connection were rejected, *n* would choose another candidate from the list and repeats the above procedure. Although this method might not find the best candidate, it avoids selecting one that is unacceptable in terms of performance.

In case that no peer is returned in the node discovery process, or all the discovered nodes have high round trip times, then n would keep original neighbor, but remains it colored as RED. The node *n* continues probing this neighbor periodically, and if the delay remains high, it would rerun PNR and try to find a replacement again. Note that if the Ring Search technique is employed, then the first non-neighbor node that responds the probe is practically the one with the smallest delay, so *n* can select the candidate in the same sequence as it receives the replies. If the GPRS and GPS method is used, the GPRS server can return the candidate list sorted in ascending order of distance. This way node *n* can use this order and try to establish connection one by one. Both of those methods do not require any knowledge of the RAON network, thus they can be served as

bootstrap mechanisms as well. However, they are both expensive operations, with ring search flooding the network, and GPRS costing the user money to access the service. Comparing to these two approaches, the Ping-Pong method is a much cheaper process, as it uses existing connections to discover new nodes, so no new connection needs to be established in advance. Therefore, RAON suggests using the Ping-Pong method to discover new peers if the node is already connected to at least one peer. For bootstrapping, GPRS is favored over ring search, since flooding is a very costly thing to do. It does not only consume bandwidth and power of the source node, but all the nodes within its TTL range. This contradicts with the main goal of the project. On the other hand, GPRS allows the mobile node to contact a server at its own expense, and no need to consume the valuable resources of other nodes. So providing that the GPRS service is accessible by the user, this method should be used rather than ring search.

The remaining question is to decide when to start the PNR process. With the NCS described in the previous section, the first requirement to trigger PNR is to have at least one neighbor that is colored RED due to high latency. Since high latency can sometimes be caused by temporary congestion or link failure, we should not drop a neighbor just because it is experiencing a short-term problem. To accommodate this, RAON nodes should record a recent history of probe results (e.g. 10 samples), and compute the average latency for each of its neighbors. These samples are also examined to see if there is an increasing pattern in latency or not. If this happens, it is likely that the node is moving farther and farther away from the source node and thus increasing the number of physical hops.

Another method to gain more accurate knowledge about the underlying network is

to introduce some new update messages for neighbors to exchange their physical locations, velocities, and traveling directions. If these data were available, they can be used in combination with the average latency to decide if PNR should be executed or not. Furthermore, rather than wasting bandwidth and power for sending regular updates, we can piggyback the mobility information to the probe messages only when needed, i.e. when latency is high. In summary, the conditions for a RAON node to replace an unstable neighbor *nbr* are:

1. *nbr* is colored RED due to high latency
2. The average latency of *nbr* is high and it shows the tendency to grow; and if *nbr* is physically far away, move in opposite direction and/or high speed (given that mobility information is accessible)

Note that PNR can be used to replace multiple neighbors concurrently, as long as they fulfill the two conditions listed above. In such cases, PNR would check if a candidate list exists or not before it launches a node discovery process. In Chapter 4, we provide the complete flow of the PNR algorithm, as well as the issues that we came across during implementation.

Finally, we do not consider replacing RED neighbor just because it is running low in energy. The main reason for replacing a neighbor is when the neighbor can affect the overall performance of a query search. Since the low energy level does not affect the existing query performance, it does not make a node candidate for neighbor replacement. However, dropping neighbor relationship with nodes running at low energy level might help conserve their energy that may prevent future query outage. It might also help the node regain its energy level by reducing the number of nodes that can forward queries to

it. This idea is discussed in the next section.

3.2.4. Energy-Aware Topology Adaptation and Flow Control

Up to this point, we have discussed how RAON nodes deal with low-resource neighbors, where both NCS and PNR operate on a per-link basis. The low-resource nodes themselves can also adapt to the environment by limiting the features it provides for other nodes. The main features of P2P file sharing applications are query search and file download. The operation of downloading a file consumes much more resources comparing to query search, since files are generally much larger than query messages in terms of size. Therefore, when a node is running low in energy, it can start its power conservation strategies by disabling the download feature first. Also, if the node is running low in bandwidth (due to congestion, serving multiple download requests, or it is downloading files from other nodes), it can temporarily stop accepting download requests as well. However, file transfers are usually done through a separate connection, thus the requester would not have any information about this node, and continue sending download requests. There are three things that the receiver can do: 1) reject the request; 2) queue the request and serve it when it regain its resources; and 3) reply the requester with a list of nodes that also possess the file. The simplest way to generate this list is to remember the nodes that have previously downloaded that file.

After the download option is disabled, if the node is still suffering from a high consumption of resources, it can also make use of topology adaptation and flow control to reduce the consumption rate. We mentioned earlier that a node assign less tokens to a neighbor that is not colored as GREEN. In fact, this can work both ways, where a low

resource node can assign smaller number of tokens to all of its neighbors.

If reducing tokens still doesn't help, the node can cut down its service even further by dropping its neighbors one by one. By reducing the number of neighbors, it does not only reduce the number of queries it would receive, but also the number of update messages. This approach is targeted for conserving energy rather than bandwidth. Due to the fact that if the low energy node drains all of its power, it would be disconnected from the network anyway, so by gradually dropping the neighbors, it might be able to extend its battery life for a little longer. When selecting which neighbor to drop, it should choose the ones that are colored as RED first, especially the ones that are low in power. The intuition of this approach is to keep as few connections as possible, but if the remaining connections are unstable or expected to be disconnected very soon, the node would have to look for replacements, which is an expensive operation as discussed before. Therefore, it should target to keep the ones that are high in power (more likely to stay in the network) with low latency (steady connection). And if the mobility information of the neighbors is known, the decision can be based on it as well, as we can expect that the nodes that are in close proximity and move in the same direction at a comparable speed (showing synchronized motion) are likely to stay connected. There should also be a minimum number of neighbors (e.g. 2) that each node must keep, so that the chance of creating network partition is smaller.

3.3. Summary

In this chapter, we have proposed RAON, a P2P system for MANET, which is based on the design of Gia. We addressed the conflicts between structured P2P systems and

MANET, and we argued that it is not suitable for MANET. We also believed that Gia might fit well in the dynamic environment of MANET. However, Gia is not specifically designed for MANET, thus issues such as link instability and limited power were not taken into account. Therefore, we modified the Gia design and added features in order to be more adaptable to the MANET environment.

We introduced NCS in RAON, where each node monitors the conditions of the links to its neighbors, as well as its neighbors' energy levels. If a neighbor is found to be unstable or running low in power, the node would avoid forwarding queries to this neighbor until the link condition improves or the neighbor recharges its battery. RAON also uses PNR that aims to restructure the overlay topology to adapt to the changes in the underlying MANET topology. In the next chapter, we discuss some of the issues that we encountered during implementation of our design.

Chapter 4 Implementation Details

The design of RAON described in Chapter 3 addresses the issues of MANET P2P systems. This chapter discusses the problems that we come across when implementing the RAON prototype. In section 4.1 we provide the approach that we used in the simulation, and then followed by a description of our prototype in section 4.2. In sections 4.3, 4.4, and 4.5, we explain the implementation details of the forwarding algorithm, the disconnect protocol, and the PNR algorithm respectively.

4.1. Packet-level Simulation

P2P system is an overlay network built at the application level. The process of simulating a P2P system on top of a network simulator with packet-level details can be very complex and time consuming. The complexity can be reduced by using detailed flow-level models instead of packet-level network model [33], which is a more feasible approach for simulating large-scale P2P systems. However, as suggested in [34], integrating a flow-level model with a P2P analytic model is in itself a complicated task. Furthermore, the packet-level details are especially important for any system that is deployed on top of MANET, since the TCP throughput can be greatly affected by the dynamic topology of the underlying network. Without the packet-level network model, it is very difficult to study how node mobility and energy consumption can influence the performance of a MANET P2P system. Therefore, we decided to use a packet-level simulator to simulate our RAON design.

4.2. Prototype Overview

We implemented both Gia and RAON using the simulation framework provided in [34]. Figure 4.1 illustrates the overview of our prototype implementation. The RAON system is divided into two modules, namely the RAON Application (RaonApp) and the RAON Agent (RaonAgent). The RaonApp is responsible for any user-level operations that include establishing connections with neighbors (topology creation and adaptation), generating queries and replies, assigning tokens (flow control), and maintaining the status of neighbors. On the other hand, the RaonAgent interacts with the TCP socket layer, implements the forwarding engine, transmits and processes messages. The rest of this section describes the functionalities of the RaonApp and RaonAgent.

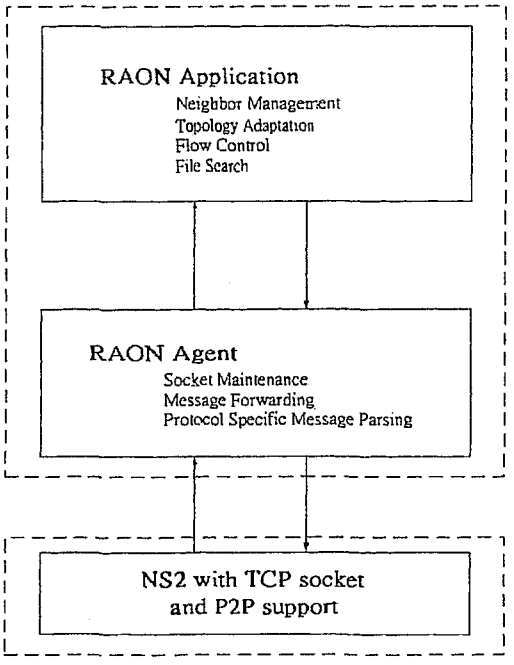


Figure 4.1: Overview of the RAON Prototype

4.2.1. RAON Application

The RaonApp maintains a table of neighbor entries, where each entry contains the following information about the neighbor: its address, capacity, degree, energy state, recent rtt history, tokens received from the neighbor, and its file indexes. When a connection between two nodes is established, each node sends initialization packets to the other node, which contains all the information about the node needed by the other node in forming the neighbor entry. In the subsequent update messages each node sends updates to the other node for any change in its degree, energy state, and file indexes. Each node also periodically sends token update messages to its neighbors every *update_interval*, and waits for the ACK packet for the purpose of computing rtt.

Topology adaptation in RAON implements the same algorithm used in Gia, except that it also keeps track of a list of neighbors that are candidates for replacement by PNR. When a node needs to drop a neighbor, for instance because it exceeds open connections with the maximum number of neighbors, it first checks whether the candidate neighbor list is empty or not. If it is not empty, it selects the neighbor with the highest average rtt value and drops it. Otherwise, it follows the algorithm of the *pick_neighbor_to_drop* function described in Gia [20] and selects a neighbor to drop.

4.2.2. RAON Agent

The RaonAgent is responsible for composing the corresponding packets and transmitting them to the targeted neighbors. Upon receiving a message, the RaonAgent first parses it and resolves the message type, and then notifies the RaonApp by calling the

corresponding API. The RaonAgent maintains a list of TCP sockets for the active neighbors and these sockets are used for all communication activities. Sockets are created and destroyed upon request by the RaonApp. The RaonAgent is also responsible for forwarding query and ping messages. The query/ping forwarding and socket maintenance are explained in the next two sections.

4.3. Forwarding Engine

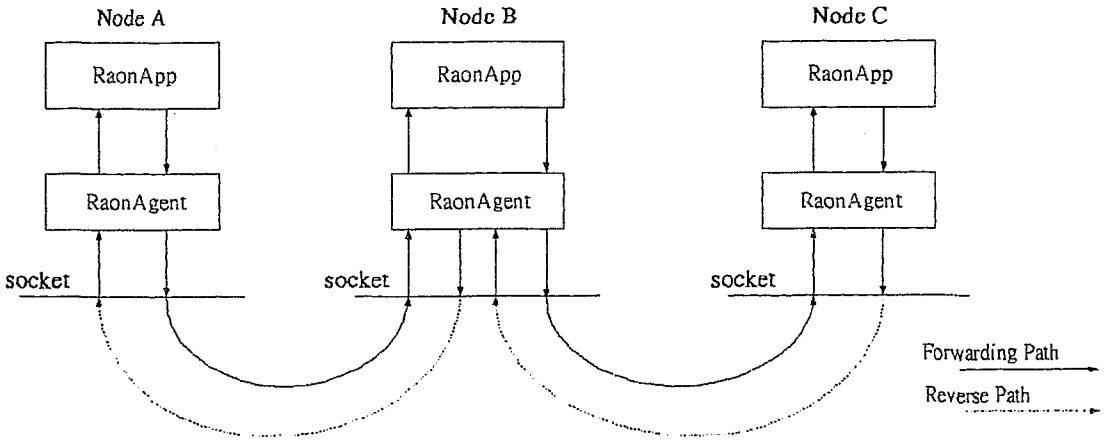


Figure 4.2: The operations involved in a query search. Node A sends a query to node B, and it forwards the query to node C. The RaonAgent of node B maintains the incoming and outgoing neighbors for each query. When node C sends back a query hit, node B can forward it back to node A.

Each RAON node performs query and ping forwarding for other peers. Figure 4.2 illustrates an example of query forwarding in a RAON network. The application in node A generates a query and passes the relevant information to the RaonAgent. The RaonAgent creates the query packet and sends it to neighbor B through the TCP socket that it has created before. When node B receives the packet its RaonAgent parses the message and determines that it is a query message. It then consults the RaonApp, which

maintains all the file indexes. If the RaonApp cannot find the requested file, it informs the RaonAgent which then forwards the query to another neighbor. The RaonAgent stores the query ID in its cache, as well as the neighbor it has received the query from. Thus a reverse path back to the query originator is established. Also, in order to avoid sending the same query to neighbors that it has previously sent to, it stores the list of outgoing neighbors for each query in its cache. The cache is periodically flushed after every *query_timeout*, which is a configurable parameter. When node C receives the query, it finds the file in its index list, so it sends back a query hit message via the reverse path (i.e. to node B). When node B receives the query hit it simply forwards the message back to node A. Ping messages also have unique identifiers and are forwarded in the same fashion as queries, except that they are forwarded along low delay paths, as described in Chapter 3.

4.4. Disconnect Protocol

The Gia implementation uses query keep-alive messages to detect for any query loss. If the query originator does not receive any keep-alive message for some time, it considers the query to be lost and generates the same query again. A query can be lost for two reasons: 1) a receiving node fails before it forwards the query to the next node; or 2) the reverse route is lost when a neighbor is dropped as a result of topology adaptation. Gia uses keep-alive messages to deal with both situations. However, it also suggests an alternative method that when a node selects a neighbor to drop the connection with the neighbor, the connection should remain open for as long as a reverse route exists that traverses the link. We implemented this method in RAON and describe the protocol

designed for this purpose in the following.

When a node X attempts to drop a neighbor Y, either due to Gia's topology adaptation or RAON's PNR, the RaonApp of X deletes Y from its neighbor list and the RaonAgent sends a DISCONNECT message to Y. The RaonAgent of X keeps the socket to Y in its socket list but changes its state to CLOSE_WAIT. Upon receiving a DISCONNECT message the RaonApp of Y deletes X from its neighbor list and changes the socket state to CLOSE_WAIT as well. Meanwhile, the RaonAgent checks if there is any pending message in its cache that uses the link to X as the reverse route. If so, it sends a DISCONN_REJ message back to X indicating that Y is not ready to close the connection, otherwise it sends a DISCONN_OK message. A node that sends out a DISCONN_OK message implies that it is ready to close the socket.

If X receives a DISCONN_OK, it checks its own cache and if the connection to Y is no longer needed, it closes the socket. If X receives a DISCONN_REJ message or it discovers that Y is still needed for a pending message in its cache, it keeps the socket state in CLOSE_WAIT state to use it for further communication. The socket in CLOSE_WAIT state remains open until the RaonAgent flushes its cache, and in that case X would send a DISCONN_OK message to Y for Y to close if it does not need the socket anymore. If Y still needs the socket it would send back a DISCONN_REJ message to X. It repeats the above process when it flushes its cache.

It is possible for a node to reclaim the connection with its neighbor after it decides to drop the neighbor and start the disconnect process. For example, node X can abolish the disconnect process with Y once it discovers that the XY link resumes stable operation. Similarly, in a different situation if X decides to replace another neighbor and discovers

that it has a good connection with Y, it is reasonable for X to reclaim its connection with Y and put Y back into its neighbor list. To reclaim the connection X sends a connection request to Y, and if Y accepts the request, both X and Y change their socket states to CONNECTED. The state diagram in Figure 4.3 summarizes the disconnect protocol.

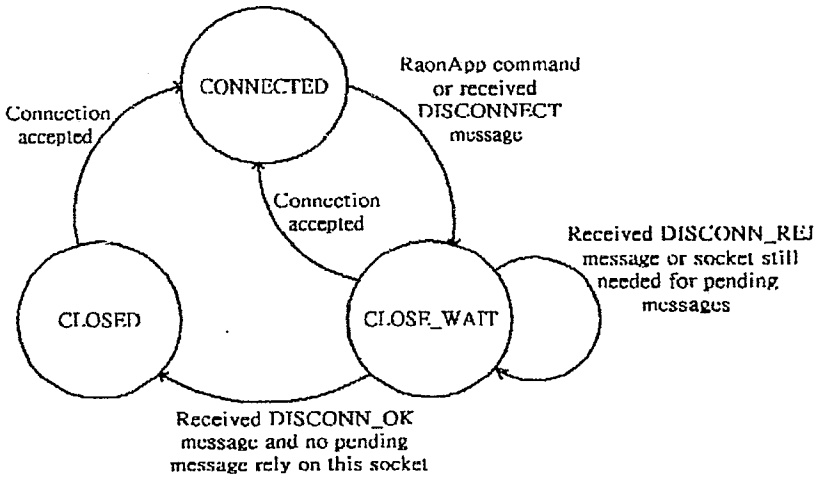


Figure 4.3: Disconnect protocol state diagram

4.5. PNR Implementation

In Chapter 3, we proposed PNR algorithm to improve the overall performance of the RAON overlay network. We also discussed there the issues that PNR is designed to solve, and described the general design of the algorithm. In this section we describe our implementation of PNR in the simulation framework.

A node in RAON employs PNR algorithm to replace a neighbor reachable through a high latency link Figure 4.4 shows the complete flow diagram of the PNR algorithm in our implementation. When a node receives an ACK from its neighbor for its prior token update message it measures the current rtt and records that in its rtt history. It then computes a new average rtt using the samples from its rtt history for that particular

neighbor. If it does not receive an ACK from the previous update, it assigns a worst-case rtt (e.g. *update_interval*) for that sample, such that it would be reflected when computing the average rtt.

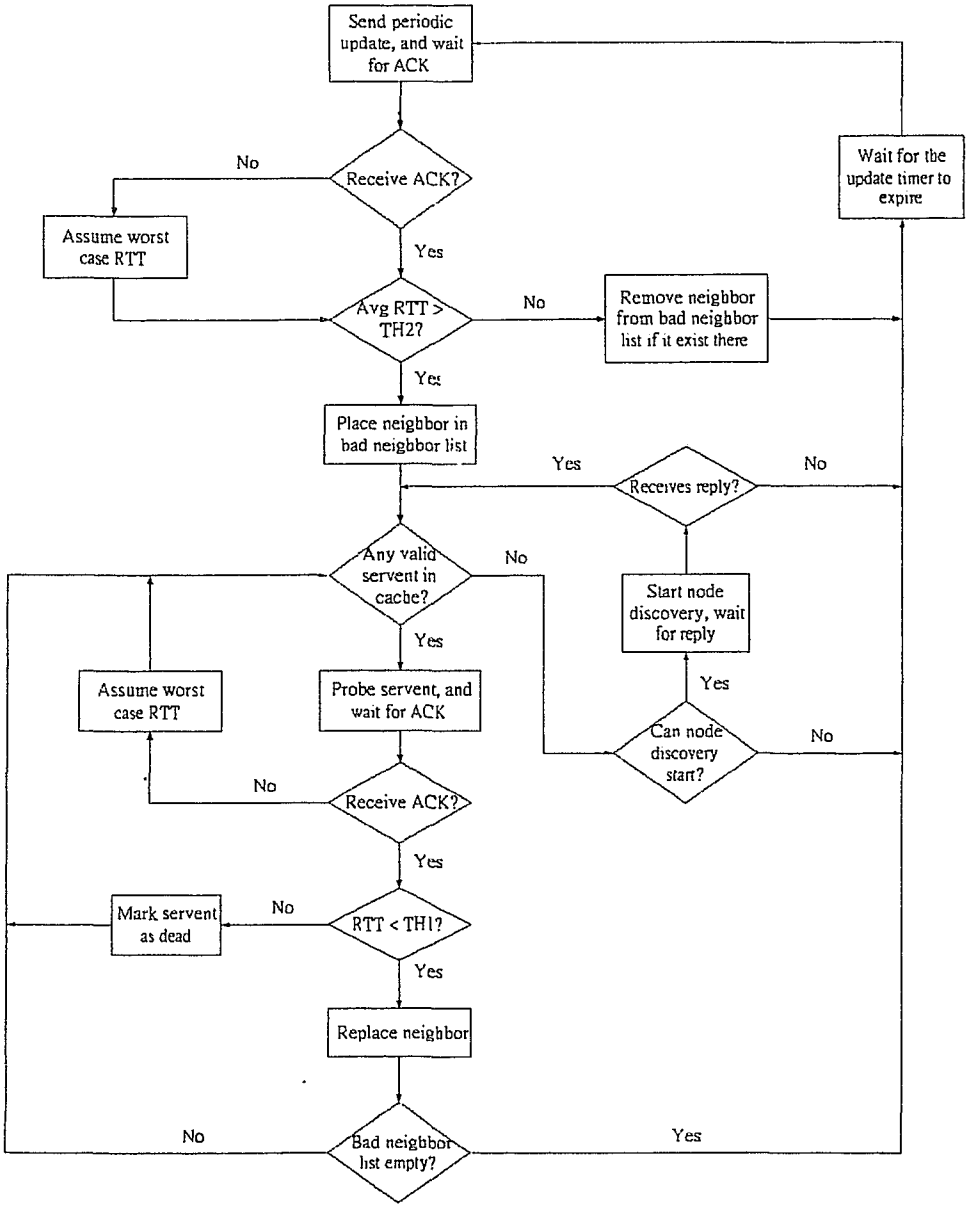


Figure 4.4: PNR flow diagram

Two predefined latency thresholds, TH1 and TH2, are used in NCS to determine the MEDIUM and HIGH latency states, respectively. The PNR also uses these thresholds to determine if a neighbor is a candidate for replacement or not. If the average rtt of a link exceeds TH2, then PNR puts the neighbor in the candidate neighbor list and start searching for a replacement. The values of the two thresholds determine the aggressiveness of NCS and PNR. In our simulation that is discussed in Chapter 5, we used the values of 1 second and 3 seconds for TH1 and TH2 respectively.

Each node has a servent cache, which is a list of known peers populated by the bootstrap mechanism or the node discovery process. The term servent is described in Chapter 2 (SERVer+cliENT), and we use it here in order to easily distinguish the peers in the cache from the peers in the neighbor list. An entry in the servent cache contains the address of the servent and the last time it was contacted. A servent is marked as dead if the node cannot contact that servent or a connection request is rejected in the previous attempt. Any servent that is not marked as dead is considered as "valid", and the PNR sends a probe message to that servent. If the node receives a reply from the servent and the average link rtt is less than TH1, it sends a connection request and tries to establish a P2P connection with the servent. If the node does not receive an ACK from the servent after a timeout, or the rtt is high (e.g. greater than TH1), or the connection request is rejected, then the node simply marks the servent as dead and tries to contact another servent.

If there is no valid servent in the cache, the node begins a node discovery process. In our implementation, the node uses the ping-pong protocol for node discovery if it has at least one good neighbor. Otherwise, it sends a request to the bootstrap server to get a

potential neighbor address. In either case, the node has a list of servents, which the PNR uses to probe the servents one by one as described above. Note that this process may run into an infinite loop if all the nodes in the servent list are already marked dead. Therefore, we added *node_disc_interval* parameter in the PNR algorithm that determines how often the node discovery process can take place. If the node has previously performed a node discovery within the past *node_disc_interval* period of time, the PNR algorithm terminates and waits for the update timer to expire.

If the node finds a servent that accepts its connection request, it then drops the neighbor with the highest delay in the candidate neighbor list and inserts the servent into its neighbor list. Recall that the purpose of PNR is to find a node reachable through a low latency link to replace a neighbor connected with a high latency link, hence the chosen node might not be the best one at that moment. The PNR then checks if there is another neighbor candidate for replacement, and repeats the above procedure as long as the candidate neighbor list is non-empty. If no more neighbors need to be replaced, then the algorithm just waits for the update timer to expire.

We suggested in the previous chapter that if a node is equipped with GPS, it may use its physical location, speed, and direction to help making the decision of dropping a neighbor as well as selecting a new neighbor. However, from our preliminary simulation results we found that the delay between two nodes has no correlation with their location and distance with each other. This is because we only have access to the physical information of the overlay nodes and it is insufficient to make a good prediction. In order to accurately predict the stability of a connection we need to have the mobility information of all the intermediate nodes. However, this is impractical and infeasible as

those nodes are supposed to be transparent to the P2P nodes. Therefore, the mobility information is not taken into account in our PNR implementation.

4.6. Summary

This chapter describes the simulation framework we used and addresses the question of why we need to do simulation with packet-level details. We provide sufficient details of our RAON prototype implementation including RaonApp, RaonAgent and forwarding engine. We also describe the disconnect protocol and the parameters to better control the PNR algorithm. In the next chapter, we evaluate the performance of RAON and compare it with Gia.

Chapter 5 Evaluation

We implemented the RAON prototype described in Chapter 3 and Chapter 4, as well as Gia using C++. We simulated both systems using NS2 and compared their performances. In this chapter, we first describe the simulation environment that we used and our simulation objectives. We then provide the results that we obtained, along with a detailed analysis and comparison.

5.1. Simulation Setup

We incorporated our RAON prototype with the Network Simulator NS2 [35]. NS2 is an open source simulator developed by the VINT project at the University of California at Berkeley, with the mobile and wireless extensions contributed by the MONARCH research group of Carnegie-Mellon University. We also installed the TCP socket and peer-to-peer extension developed by the Networking and Telecomm Group at Georgia Institute of Technology. Our simulations are based on NS2 version 2.26 installed on Redhat Linux 9. We use the IEEE 802.11 standard MAC layer. Each node is equipped with an antenna that is 1.5m above the ground. We use the two-way ground propagation model, with a transmission power of 281.8mW, which results in a transmission range of approximately 250m. The energy model provided by NS2 is used to model the node's energy consumption and all nodes start with the same initial energy level. Node movements are modeled with the random waypoint mobility model, where each node chooses a destination randomly within the simulation area using a uniform random speed between 0 and *max_speed*. When the node reaches the destination, it pauses for a fixed

amount of *pause_time*, which is set to 30 seconds in all the scenarios, and then move on to another destination.

Our simulation objective is to study the behavior of RAON and Gia and evaluate their performances under different system configurations and network conditions. We simulated two RAON configurations. First, we refer to RAON the system implementing both NCS and PNR. Second, we refer to (RAON – PNR) the system implementing only NCS but not PNR. By varying the node density and the maximum speed of mobile nodes, we generated MANET scenarios with various levels of dynamism. Table 5.1 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. For node density, we keep the number of mobile nodes constant, and use two different simulation areas; (500m x 500m) and (1000m x 1000m), to achieve dense and sparse node distribution respectively. In the rest of this chapter, we use the notion *max_speed:dimension* to refer to a scenario, where *dimension* is the dimension of the simulation area (i.e. 500 and 1000). Thus, we generated six scenarios for all combinations of speed and dimension. For example, 2:500, 5:500, 20:500, 2:1000, 5:1000, 20:1000 are scenarios 1 to 6 respectively.

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

Table 5.1: Network level simulation parameters

The simulation begins with a defined MANET topology where the nodes are randomly placed over the simulation area. We used AODV as the underlying routing protocol in MANET [8]. Our system implementation is independent of the underlying protocol and should work with any routing protocol. However, its performance may differ, which is an area of further investigation. We chose AODV because it is an on-demand routing protocol with local route repair, which is known for exhibiting superior performance among protocols of different classes [36]. At the P2P level, each overlay node uses a uniform random number between [0, *sim_time*/2] to determine when to join the network. Therefore, no P2P topology is defined in the beginning of the simulation. A virtual bootstrap server exists in the simulation that models the GPRS bootstrap approach described in section 3.3.2. The server contains a complete list of existing P2P nodes, so new nodes can learn about other P2P nodes by contacting it. After a node is connected to the overlay network, it periodically exchanges with its neighbors update messages every *update_interval.* Since the AODV implementation in NS2 uses a value of 60 seconds as the lifetime for each discovered route, we decided to set our *update_interval* to be 30 seconds.

Queries are generated at a minimum rate of 1 query per 10 seconds to model the behavior of aggressive P2P users. Each query contains a keyword which is represented by an integer between [0, 99], and every keyword is mapped to a set of files. The files located on each node are generated randomly, but the number of files and the file sizes are dependent on the node's capacity level. This is to reflect that different wireless devices have different levels of resources, particularly storage space. However, in our

simulation, we use the same capacity level for every node. It is because we want to focus on the effect of different MANET scenarios have on the performance of the P2P systems, thus we need to limit the number of variables in our simulation in order to have a better understanding. Table 5.2 summarizes all the simulation parameters used at the P2P level.

Parameters	Values
Number of peers	10, 20, 30, 40, 50
Start Time	Uniform random between [0, 500]
Query Generation Interval	Uniform random between [0, 10]
Number of files stored at each peer	Uniform random between [0, 80]
update_interval	30 sec
query_timeout	240 sec
Latency thresholds *TH1*, *TH2*	1 sec, 3 sec
Energy thresholds *TH1*, *TH2*	50%, 20%
max_responses	1
TTL	32

Table 5.2: P2P level simulation parameters

5.2. P2P Query Forwarding Performance

Both Gia and RAON use biased random walk to forward queries. The critical difference between Gia and RAON is that the capacity level of a neighbor is the only factor that biases the random walk in Gia. On the other hand, link color is also a factor biasing the random walk in RAON, which is expected to improve the quality of forwarding decision and as a consequence improves the query success rates and query delays. The main performance metrics we used for evaluating the relative query forwarding performance of Gia and RAON are query success rates and query delays. But before discussing those measures we present the evidence of quality of forwarding

decisions. In order to compare the quality of forwarding decisions, we implemented Gia in a way such that each node monitors its rtt with its neighbors as in RAON. The only difference is that Gia would not take the neighbor's rtt into consideration when forwarding queries. Thus, we can compare the quality of the links used by both systems.

Recall that query responses are forwarded back to the originator via the reverse path. Thus, if a query is sent using a green link, it is probable that when the response is routed back to the query source, the link has changed its color to yellow or red. Therefore, we have traced the link colors separately for forward and reverse paths. We then computed the percentage of links used for each color. Since we want to compare the links chosen by RAON to the ones chosen by Gia, we plotted the link percentage differences relative to RAON for the average of all scenarios in Figure 5.1. This means that the curves in the figure are computed by taking the percent of each color used in RAON and then minus the percent of the corresponding color used in Gia. For example, the curve labeled "Forward Green" in the graph means that the percentage of green links used for forwarding in RAON is that amount more than Gia. And intuitively, a negative value means Gia is using more links of that color than RAON. We also like to distinguish the red links that are high delay and red links that are low energy, so we differentiated the two by using red and pink curves in the graphs, which are labeled as Red(d) and Red(e) respectively. Finally, the solid curves show the percentage of a particular colored link being used for both forward and reverse paths.

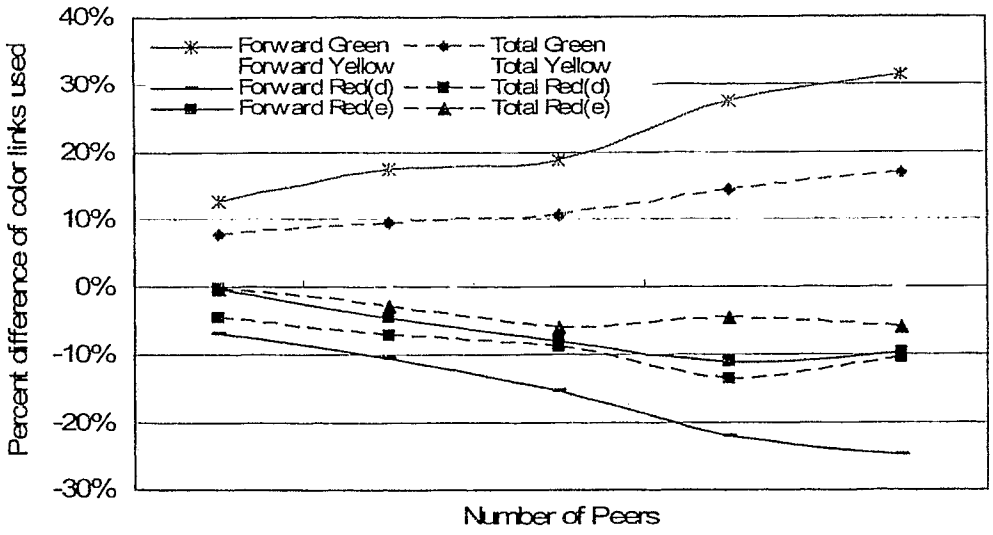


Figure 5.1: Percent difference in colored links used by Gia and RAON for the 20:1000 scenario

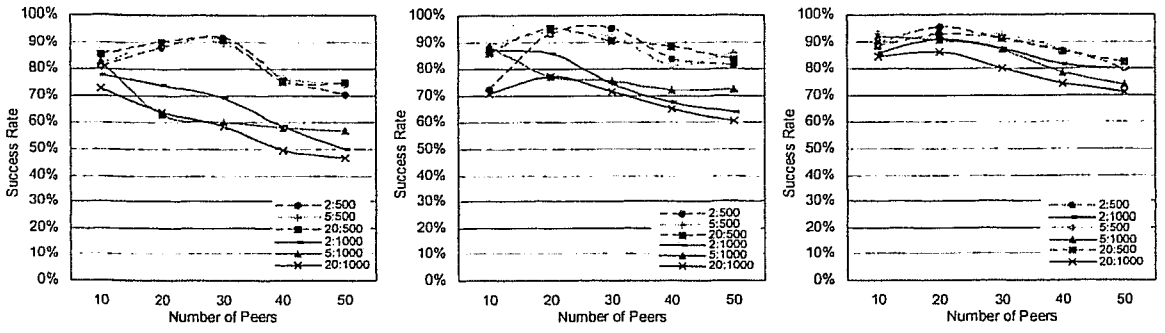
The above graph illustrates that in general RAON tends to use more green links than Gia along the forward path. Another expected trend is that Gia uses more red links than RAON, as expected. However, quite unexpectedly, the total number of red links used in RAON is rather close to Gia. This indicates that RAON uses more red links for reverse routes than Gia. Furthermore, since NCS only monitors the link every *update_interval*, the node continues using the link as green and preferring this link over the others even if the link latency increases shortly after the link is marked green. Although we can decrease the value of *update_interval* to achieve better accuracy of the link conditions, update messages are control messages, so they are part of the protocol overhead. By sending update messages more frequently, this means introducing more overhead to the protocol and also consumes more energy. Therefore, more investigation is needed in order to select a good value for *update_interval*. We omitted the color link graph for (RAON – PNR) here since it is very similar to the RAON one. This is because

forwarding decisions are solely based on NCS, which is implemented in both RAON and (RAON – PNR). The actual color link graph for each system is provided in Appendix A.

To evaluate the performance of a P2P system, we measure the success rate of query search and the query delay. A query is considered successful if the query originator receives at least one query hit response before the query timed out, and the query delay is the time it takes for the originator to receive the query hit response. The overlay hop count, which is the number of overlay hops that the query travels before finding a match, is typically used as a performance measure [20]. Since the size of our network is much smaller than the one used in Gia (50 comparing to 10,000), we found average overlay hop counts that range from 1 to 1.12. This also shows that the files are well-replicated over the network, since most queries are able to find a match with only one hop. Therefore, we do not consider overlay hop count as one of our performance metrics. We first examine the performance of Gia in different MANET environment, and then we compare Gia, (RAON – PNR), and RAON under the same circumstances. In the next section, we analyze the underlying network behavior and the impact it has on the P2P performance of both systems.

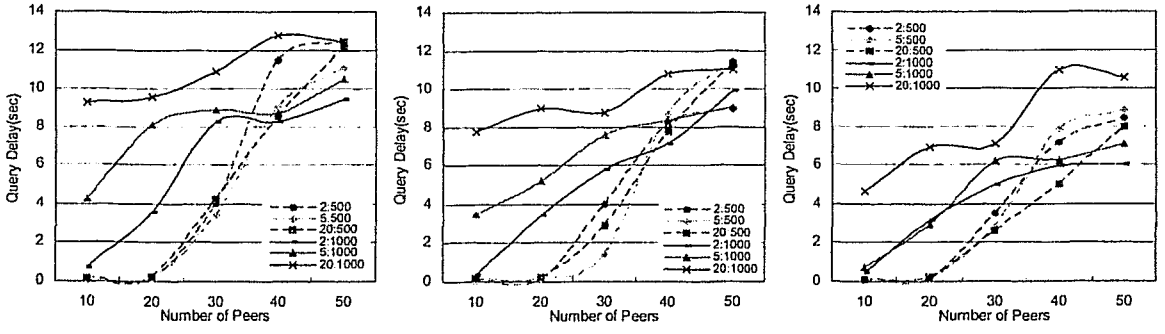
We measure the query success rate and query delay with increasing number of peers. The results of those two metrics are plotted in Figure 5.2 and Figure 5.3 for Gia, (RAON – PNR) and RAON, we can make following observations about the results. First, we notice that as the number of peers increases, the success rate in Gia tends to drop while the query delay shows a rising trend. By increasing the number of peers, the network becomes more unstable, which increases the chance of query loss, and

consequently drops the query success rate. The network traffic also increases with the number of peers, which increases the traffic load on every node and causes higher query delay. We can visualize this situation by recalling Figure 3.2, where the mobile node closest to node 2 is in fact responsible for all three P2P connections. Therefore, if that node is congested, then any query that originates or destines at node 2 is at risk of being dropped.



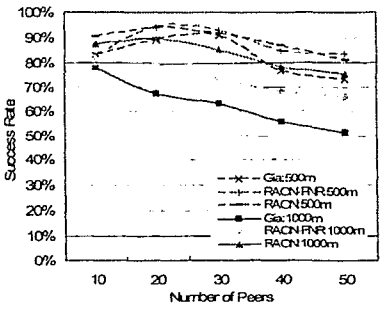
(a) Gia (b) RAON – PNR (c) RAON

Figure 5.2: Query Success Rate for Gia, (RAON – PNR), and RAON under different MANET configurations

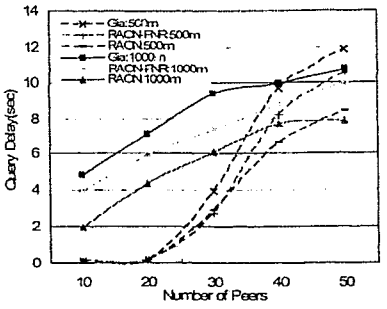


(a) Gia (b) RAON – PNR (c) RAON

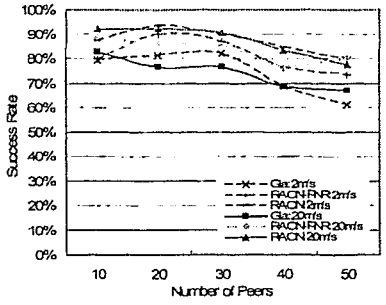
Figure 5.3: Query delay for Gia, (RAON – PNR) and RAON under different MANET configurations



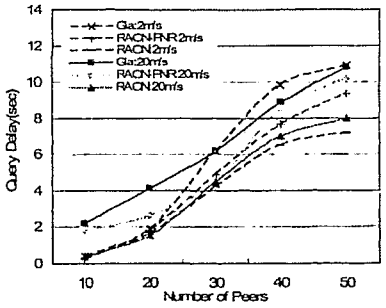
(a) Average query success rate of various speeds



(b) Average query delay of various speeds



(c) Average query success rate of various dimensions



(d) Average query delay of various dimensions

Figure 5.4: Average query success rate and query delay by keeping the simulation area constant in (a) and (b), while keeping the node speed constant in (c) and (d).

Second, by increasing the simulation area and the node's maximum speed, the performance degrades. However, the impact of node density on the relative performance of Gia, RAON-PNR and RAON is noticeably higher than node speed. This observation can be further confirmed in Figure 5.4. In Figure 5.4(a) and (b), we computed the average query success rate and query delay respectively over all three speeds for both 500m and 1000m network dimensions. Likewise, we computed the average query success rate and query delay over the two dimensions for all three speeds. However, for clarity we omitted the graphs for 5m/s as they do not show any different trend and showed the graphs for 2m/s and 20m/s in Figure 5.4(c) and Figure 5.4 (d). We

can observe from Figure 5.4 that the variation in average query success rates and query delays are more pronounced for different network dimensions as compared to different node speeds. Therefore, we can conclude that node density has a higher impact on the performance than node speed.

Third, for the (500m x 500m) scenarios in Figure 5.4 the gaps between the average query delays of the three systems increase with number of peers. This is less pronounced for sparse network, i.e. (1000m x 1000m). Increasing number of peers in a network of given size increases the network activities, which in turn increases the chance of congestion. It seems that the congestion is a major cause of the gaps in delay than the node mobility, because it is more pronounced in a dense network for high number of peer nodes where mobility has smaller impact. This is further reinforced by the fact that the average query delays for (500mx500m) become comparable with (1000mx1000m) as the number of peers increases. In (1000mx1000m) both congestion and node mobility cause increase in the query delays.

Fourth, RAON exhibits low variation in query success rate with increasing number of peers as compared to Gia as shown in Figure 5.2. For example in the highly dynamic scenario of 20:1000, the query success rate for Gia varies from 72% to 48% whereas for RAON it varies from 85% to 71%. The low variation in RAON is primarily due to the quality of its forwarding decision. We have observed that the average overlay hop count for both Gia and RAON is 1-2 hops. It means that the forwarding decisions made at the source node and by intermediate nodes along the query path have impact on the success rate. In case of RAON the source does not generate queries unless it finds a good link. In our discussion the good link is either green or yellow and the bad link refers to a red

link. Holding a query for a good link by the source may cause delay in query generation time but it contributes to increase the chance of query success. When a node selects a good link to forward a query it increases the chance of query hit because: (1) the chance of query being dropped along the path to the next hop is low as it follows a stable route to the next hop, and (2) it is more probable for the next hop to process and if necessary forward the query instead of dropping the query as it has higher residual power to do so. However, since RAON tends to use good links only in the forwarding path and does not guarantee that when the reverse path uses the same links they remain good links, we see some drop in the query success rate. We also realize that if PNR is disabled (i.e. for (RAON – PNR), NCS alone can still achieve better performance than Gia in terms of query delay and success rate, however the pattern is not as stable as RAON that implements PNR.

For query delay as shown in Figure 5.3, the RAON manages to have a slight performance edge over Gia mainly due to the quality of its forwarding decision. However, by choosing a green link for the forwarding does not necessarily yield a low delay because of no guarantee of good links along the reverse path. The average delay of 10 seconds is still fairly high. It seems that RAON suffers similar performance hit due to congestion and node mobility as Gia. In order to obtain more meaningful results, we plot a histogram based on the query delays in Figure 5.5 for the 20:1000 with 50 peers in RAON, and we discover that more than 40% of the query responses are received within 100ms, and about 80% of the queries are received less than 5s. The average delay shown in Figure 5.3 is largely affected by some query delays that are unacceptably high (more than 3 minutes).

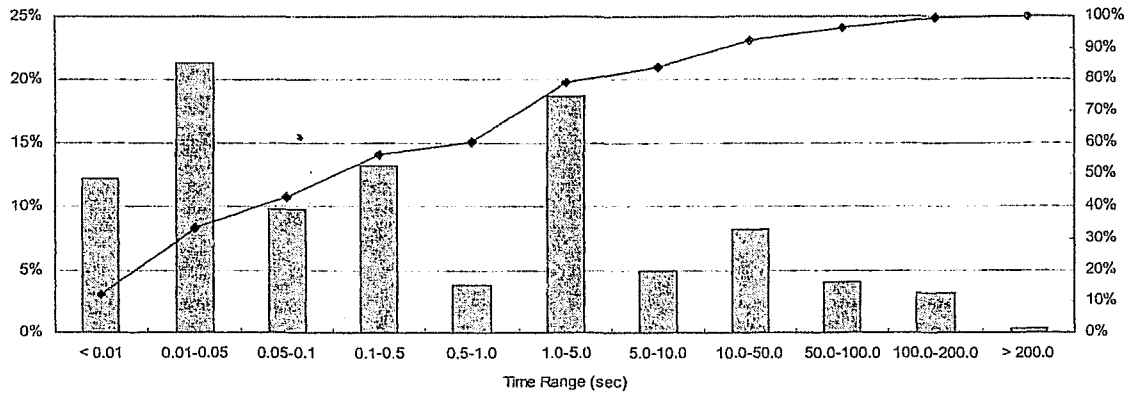
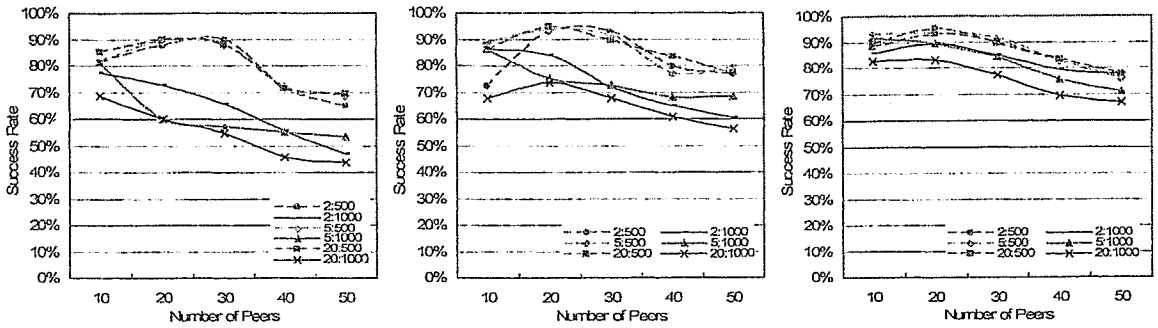


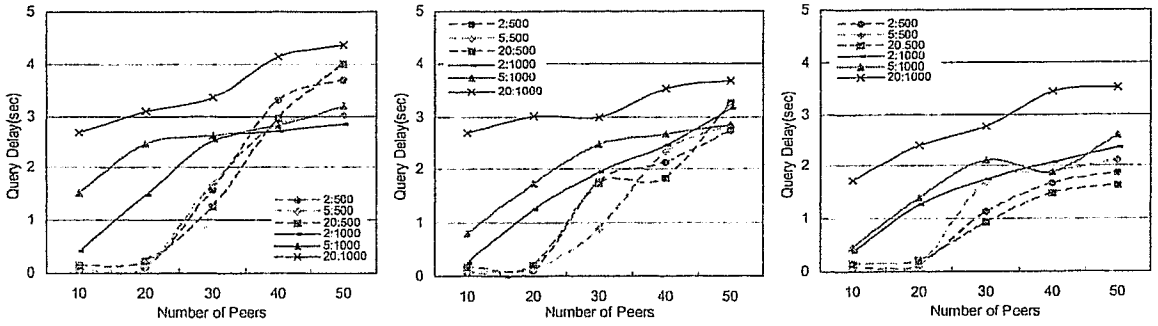
Figure 5.5: Histogram (bar) and CDF (line) of query delay distribution

Practically, a user does not tend to wait too long for a query response. In fact, he or she would think that no match is found and simply initiate another search with a new keyword. Therefore, we decide that those high delay entries should be counted as query misses instead of hits. We determine that a user usually does not have the patience to wait for a response for too long, thus we exclude any query hit entry with a delay more than 60 seconds. This is in fact equivalent to changing the value *query_timeout* in the simulation. The resulting graphs after counting high delay query as misses are shown in Figure 5.6 and Figure 5.7 both for query success rate and query delay respectively. As expected, the query success rate dropped slightly for all three systems, since we have defined more query misses. However, RAON still manages to maintain a relatively stable success rate than the other two systems when P2P network size changes, due to the same reason discussed before. Meanwhile, the query delay drops dramatically for the three systems, and RAON again has a slight advantage over Gia.



(a) Gia (b) RAON – PNR (c) RAON

Figure 5.6: Query Success Rate for Gia, (RAON – PNR), and RAON under different system configurations and network conditions after excluding unacceptably high delay entries



(a) Gia (b) RAON – PNR (c) RAON

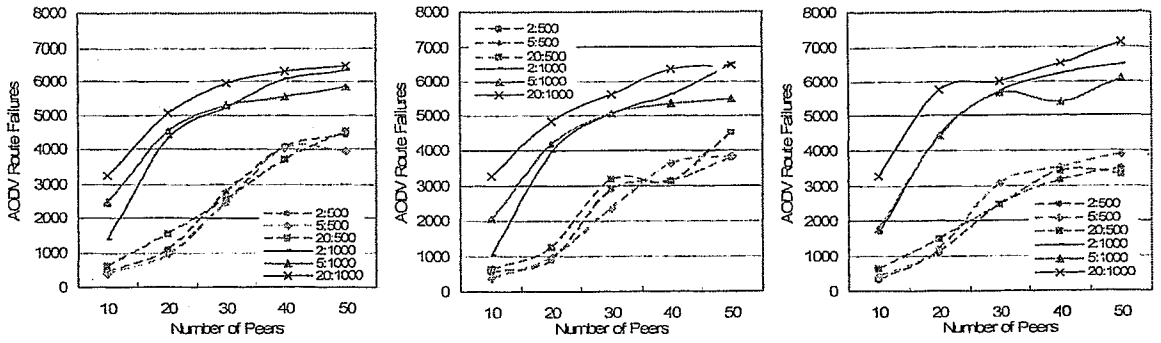
Figure 5.7: Query delay for Gia, (RAON – PNR) and RAON under different system configurations and network conditions after excluding unacceptably high delay entries

5.3. Network-level Analysis

In the previous section, we have evaluated and analyzed the query forwarding performance of both Gia and RAON using the data collected at the P2P level. In this section, we evaluate their performance by examining the network-level details. A virtual link between two overlay nodes is usually formed by a path consisting of multiple

physical hops in the underlying network. If a single mobile node along the path moves from its current position, the virtual link (route) may break and causes AODV to resort to a recovery process either by invoking local repair mechanism or simply sending a notification to all upstream nodes along the route, as described in Chapter 2.

Figure 5.8 shows the statistics of the AODV route failures that the network experiences in the simulation for Gia, (RAON – PNR), and RAON. The graphs show that all three systems suffer approximately the same number of route failures in all MANET scenarios. We notice that as the number of peers increases, the number of route failures increases as well. This is because: (1) the latent failures are exposed, and (2) the number of query forwarding paths using a failed link increases. We explain the two situations one by one using an example P2P network shown in Figure 5.9 where the shaded nodes (A-C) are P2P nodes and two link failures happen in the underlying network. First, a link failure remains latent if the link is not along any query forwarding path. For example, the link failure LF1 remains latent as long as node 1 does not become a P2P node. Therefore, as the number of peers increases (e.g. node 1 becoming a P2P node), more routes are used for query forwarding, which consequently expose more latent link failures. Second, the use of failed links increase for query forwarding as the number of peers increases. For instance, the link failure LF2 was along the query forwarding paths of AB and AC in Figure 5.9 before node 1 has joined the P2P network. As node 1 becomes a P2P node the use of LF2 for query forwarding has increased as it now lies along the forwarding paths of AB, BC, 1B and 1C.



(a) Gia (b) RAON – PNR (c) RAON

Figure 5.8: AODV Route Failures for Gia, (RAON – PNR) and RAON under different system configurations and network conditions

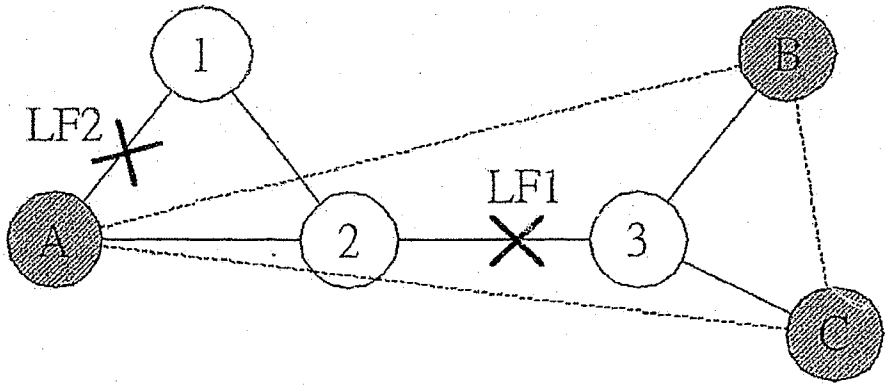
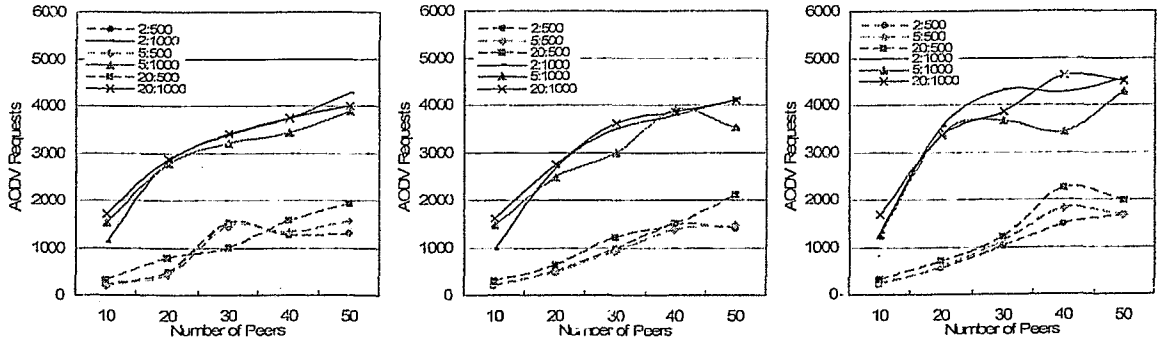


Figure 5.9: Link failure example. The dotted lines represent virtual links, and I F1 and LF2 are link failures 1 and 2 respectively.

Route failures may lead to congesting a node. This happens when an intermediate node along the route detects a link failure; it starts the local repair process if it is closer to the destination than the source. During the local repair process, packets targeted for the destination are queued at the node performing the local repair. Due to excessive delay of the queued packet source TCP may experience time out causing retransmissions. When the intermediate node finds a new path, it flushes all the queued data along the new route, and thus the destination receives duplicate packets. Furthermore, packets are

dropped if the queue is full, which also causes retransmissions.

We plotted in Figure 5.10 the number of AODV route requests during the simulation lifetime. From the graphs, we notice that generally RAON generates slightly more route requests than Gia and (RAON – PNR), indicating that it explores more routes in the underlying network. However, since AODV uses broadcast to discover a new route, the small increase of route requests generates a large number of route request messages in the network. Table 5.3 summarizes the average number of messages each system generates in all MANET scenarios. It is worth noting that the number of messages generated in (RAON – PNR) is comparable to Gia, which implies that NCS alone does not create any significant overhead. On the other hand, RAON that implements PNR generates probe messages to discover new neighbors, resulting in close to 5000 more AODV request messages than Gia on average. This indicates that RAON introduces more overhead and most of them are caused by PNR, particularly for exploring routes to the neighbor candidates.



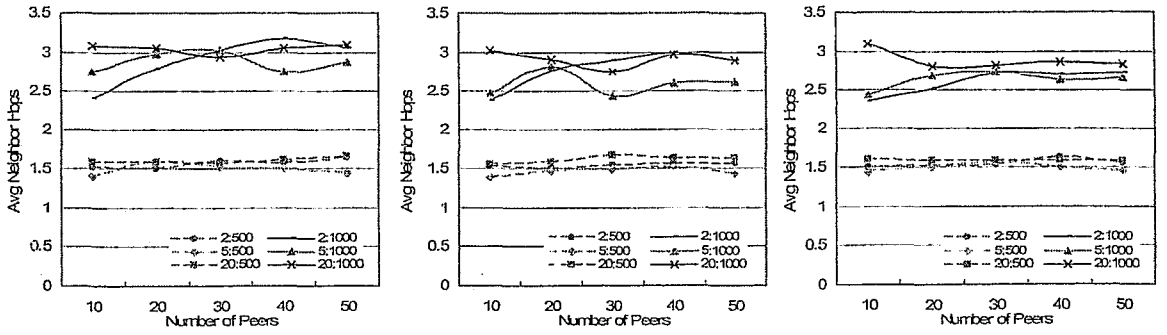
(a) Gia (b) RAON – PNR (c) RAON

Figure 5.10: AODV Requests for Gia, (RAON – PNR) and RAON under different system configurations and network conditions

Type of Message	Gia	RAON – PNR	RAON
All forwarding messages (Query, Query Hit, Ping, and Pong)	10187.48	9504.60	7887.06
Token Updates + ACK	3886.42	4010.72	4084.76
Connection	161.54	166.57	159.82
Probe + ACK	0	0	87.72
Retransmissions (Token Update and Query Hit)	1652.85	1582.40	1318.24
AODV request messages	33860.61	33947.09	38622.23
Total	49748.9	49392.66	52159.82

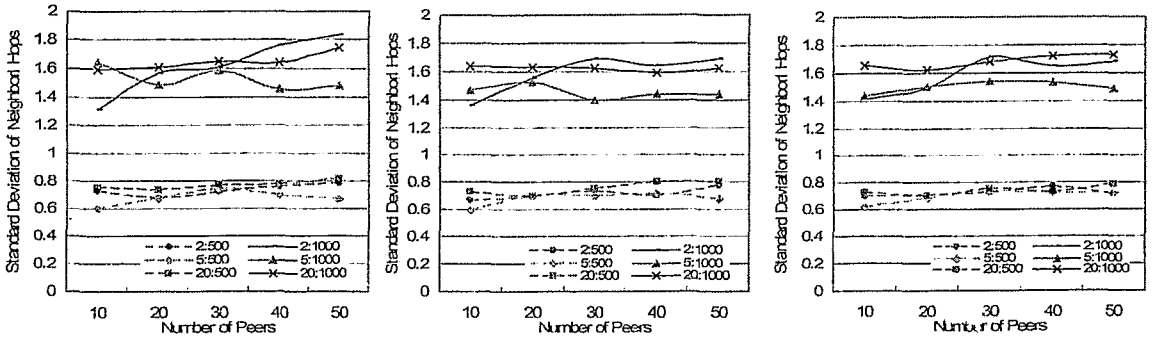
Table 5.3: Average number of messages generated over all MANET scenarios for Gia, (RAON – PNR), and PNR.

We also extracted information on physical hop counts to reflect the neighbor relationship. Figure 5.11 illustrates the average hop count between two neighbors, and it shows that this value is dependent on neither the number of peers nor the node speed. Rather, it is mainly affected by the node density of the area. Figure 5.12 shows the standard deviation of the neighbor hop counts and again it is independent on the P2P network size as well. In order to find out the worst case scenarios, we plotted the maximum number of hops between two neighbors in Figure 5.13. It shows that some of the neighbors are 20 or more hops away from each other. Connections that involve too many physical hops do not only consume more bandwidth and energy of the network as a whole, they also increase the chance of encountering link failures. Therefore, we believe that if we were able to gain access to the routing tables and learn about the physical hop count to each neighbor, we can use this as supplementary information to make better decisions in NCS and PNR.



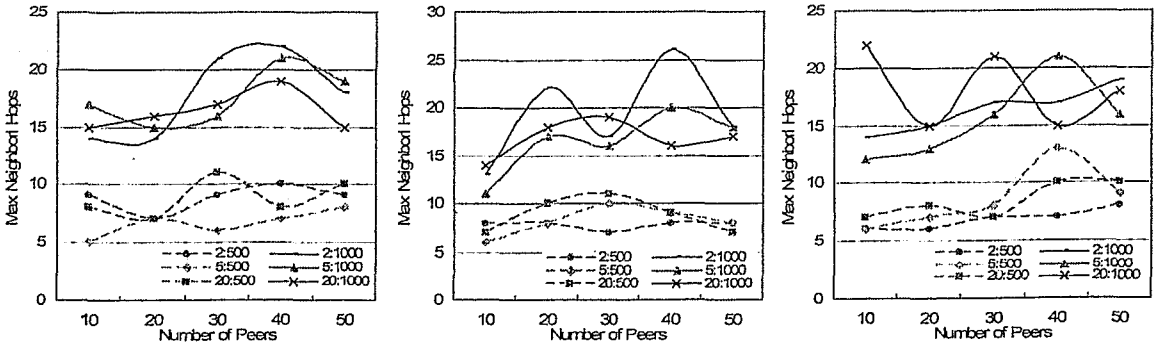
(a) Gia (b) RAON – PNR (c) RAON

Figure 5.11: Average physical hop counts between the neighbors for Gia, (RAON – PNR) and RAON under different system configurations and network conditions



(a) Gia (b) RAON – PNR (c) RAON

Figure 5.12: Standard deviation of physical hop counts between the neighbors for Gia, (RAON – PNR) and RAON under different system configurations and network conditions

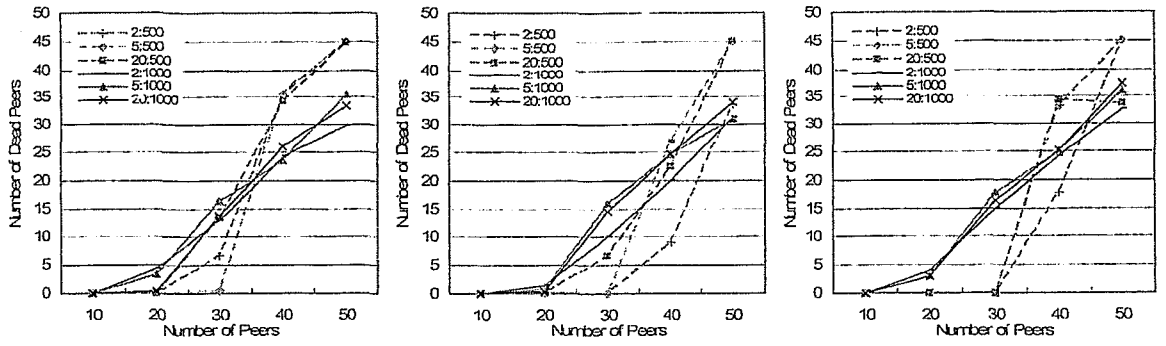


(a) Gia (b) RAON – PNR (c) RAON

Figure 5.13: Maximum physical hop counts between the neighbors for Gia, (RAON – PNR) and RAON under different system configurations and network conditions

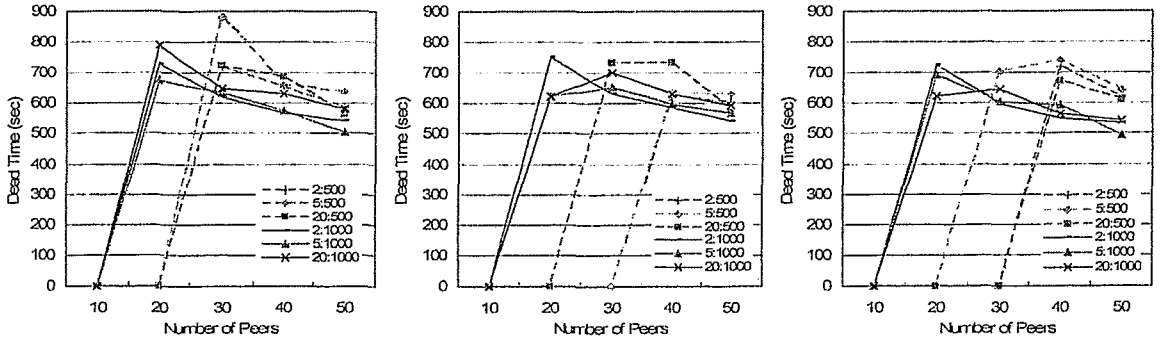
...ergy Consumption

One of our design objectives is to conserve energy consumption. However, from the simulation results, we realize that RAON is in fact consuming more energy than Gia, and also creating more dead nodes. A dead node is a node that runs out of power and failed before the simulation ends. Figure 5.14 shows the number of dead nodes, and Figure 5.15 shows the average time it takes for a node to run out of power (dead time). The graphs show that the energy consumption rates for all three systems are approximately the same. However, as shown in Table 5.3, the average number of forwarding messages generated in each system is different (with RAON generating the least), and all the other messages are simply overheads. So in Figure 5.16, we normalize the dead time by the number of forwarding messages and it shows that RAON generally runs out of power much faster than the other two systems, which implies that it consumes more energy. The only feature that would make RAON consumes more power is PNR, and with the results of more AODV route request messages being generated as shown in the previous section, we believe that our PNR implementation is too aggressive and needs to be modified. We can control the aggressiveness of PNR by changing either the rtt history size (such that PNR is not easily triggered by a small number of high rtt samples) or the values of the delay thresholds. However, a better approach might be to gather more accurate information of the underlying network condition, such as to gain access to the routing tables, as suggested in the previous section.



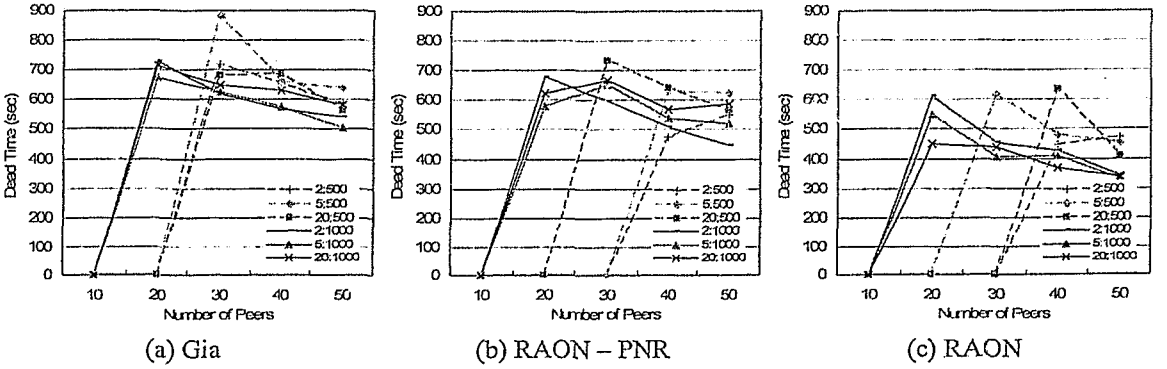
(a) Gia (b) RAON – PNR (c) RAON

Figure 5.14: Number of dead nodes for Gia, (RAON – PNR) and RAON under different system configurations and network conditions



(a) Gia (b) RAON – PNR (c) RAON

Figure 5.15: Dead time for Gia, (RAON – PNR) and RAON under different system configurations and network conditions



(a) Gia (b) RAON – PNR (c) RAON

Figure 5.16: Normalized dead time for Gia, (RAON – PNR) and RAON under different system configurations and network conditions

We also plotted the energy consumption pattern of each individual node in the 20:1000 case with 50 RAON peers, which is shown in Figure 5.17. From the graph, we notice that most of the nodes experience a linear consumption rate. We then isolate one node that does not experience quite a linear rate (see Figure 5.18), and notice that the node joins the overlay network at around 500 seconds but by that time it has already consumed 50% of the energy. This indicates that although a node does not generate overlay traffic or forwarding any queries, it still consumes more power in RAON comparable to other overlay nodes as it carries control messages. We also observed that the node's consumption rate slows down towards the end of the simulation time and it remains alive, because as many nodes started to run out of power, the size of the overlay network decreases resulting in lesser traffic flowing in the network. We suggested in Chapter 3 that a node can save its energy by assigning lesser tokens to its neighbors, and even dropping some neighbors in order to decrease the traffic flowing through the node. However, the above observation shows that the energy saving scheme as proposed in Chapter 3 has little benefit in saving energy as the traffic in the underlying can still drain power of the overlay node.

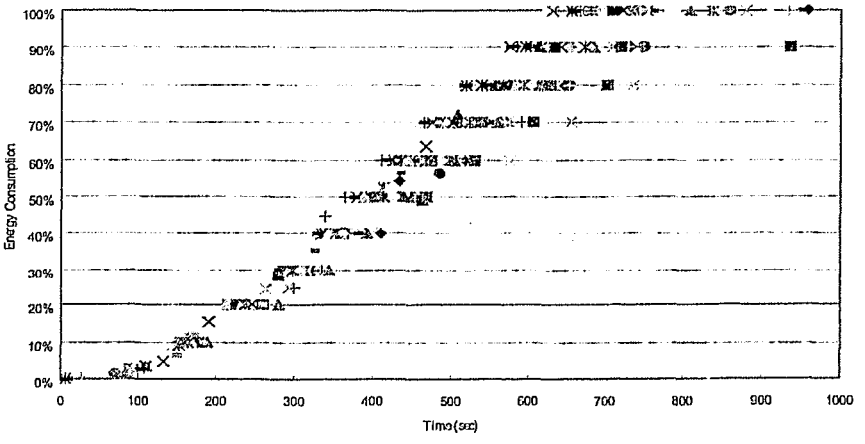


Figure 5.17: Energy consumption pattern for all RAON nodes in the 20:1000 scenario with 50 peers

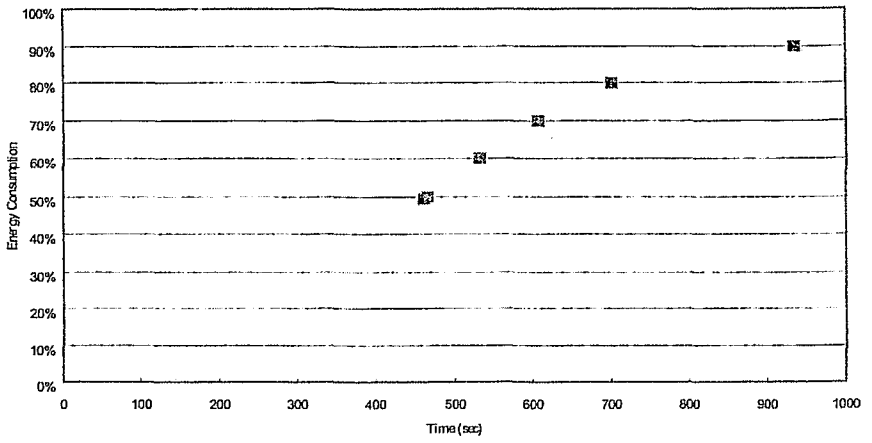


Figure 5.18: Isolating the energy consumption of one of the nodes in Figure 5.12

Another issue that concerns energy consumption is retransmission. TCP retransmits packets that are not acknowledged after a timeout. This happens even if the route to the destination is repairing or it no longer exists. Energy consumed in such cases is unnecessary. If we could reduce the number of retransmissions, it may help to lower the overall energy consumption of the network. We discuss this in more details in Section 5.6.

5.5. PNR Overhead

In Section 5.3 we discussed that RAON initiates more AODV route requests and consequently generate more AODV route request messages primarily due to the PNR's tendency of exploring and using significant number of more routes than Gia. Another cost that we must consider is the high level of energy consumption by PNR, which is discussed in Section 5.4. Hence, the PNR algorithm should be treated as a

supplementary scheme for RAON and the decision of when and how to use it should be carefully evaluated considering its associated overhead.

5.6. MANET P2P Open Questions

The results we presented in this chapter give us a much better understanding on MANET P2P systems. The observations that we made here lead us to three open questions:

1. Reverse path for responses?
2. TCP or UDP?
3. How effective is application level Flow Control?

In this section, we discuss each of these questions, and provide our thoughts based on our simulation results.

5.6.1. Reverse path for responses

From our simulation results, we notice that RAON experiences very high delay in some of the queries, and most of them are due to unstable reverse paths. A link that is marked green in the forward direction does not necessarily mean it is green in the reverse direction as well. Also, it is possible that the node that generates the response (the response node) can find a more stable route to the source than the one recorded by the forwarding nodes. Therefore, we may reduce the query delay by sending replies directly from the response node back to the query originator.

The problem with this approach is that the response node usually does not have an existing route to the query source, and in such cases it must go through the route discovery procedure first. Also, if the reverse path is in fact very stable, this method may seem redundant since it is trying to discover a route that already exists. Furthermore, as the simulation results show that, if the number of peers is small (e.g. 10), the query delay is generally very low. Therefore, there are situations where the reverse path is better, while there are other situations where setting up a new route may benefit more. We may design a hybrid approach that can take advantage of the two methods, and we elaborate on this idea in the next section.

5.6.2. TCP or UDP

kAC.N nodes are connected using TCP connections. The choice of using TCP is based on the fact that it is a reliable transport protocol, which helps us to reduce the risk of losing a query or update message. However, under the dynamic environment of MANET, a reliable protocol like TCP may become unreliable as well. Our simulation results have shown that, as the number of peers increases, the network becomes unstable. In such cases, retransmissions caused by TCP create even more data flows to the network that is already suffering from high traffic load.

Instead of TCP, maybe we can use UDP and implement retransmission support at the application level, which gives complete control to the application on whether retransmission should occur and how often. This approach has three advantages. First, we can set the maximum number of times a node can retransmit a message, and we can further define a different maximum number for different messages. Secondly, when a

node is forwarding a query, it chooses the neighbor with the highest color known at that moment. If it turns out that the selected neighbor is no longer in expected good condition, it can choose a different neighbor and continue forwarding. The node can even lower the color of the previously selected neighbor such that it would not treat that neighbor as a high color neighbor anymore.

In the previous section, we suggested to use a hybrid approach for routing back responses to the query source. If we use UDP as the transport protocol instead of TCP, it may help us to accomplish this task. This is because with UDP, the retransmission decision now shifts to the application level. If an overlay node along the query reverse path realizes that the link to the next (overlay) hop become unstable (i.e. requires retransmission), then the node can try to establish a route to the query source and send the response directly. If a route cannot be found, the node would queue the response locally and repeats the above sequence (reverse path then direct route) again later. The major disadvantage of using the UDP approach is that it increases the application developer's responsibility.

5.6.3. Effectiveness of Overlay Flow Control

Gia implements a flow control mechanism in order to avoid overloading a node with queries. However, flow control that is implemented at the application level is not sufficient for a MANET node. In a wired network, all P2P nodes are assumed to be end-hosts, where they do not need to do any routing. This is obviously not the case for MANET nodes. As illustrated in section 5.4, the energy consumption rate of a non-P2P node is comparable to a P2P node. Therefore, even if a P2P node assign very little

tokens to its neighbors, it may still experience congestion due to high underlying traffic. Hence, flow control support at a lower level maybe desired.

5.7. Summary

In this chapter, we presented the simulation results of Gia and RAON under certain MANET scenarios. We discovered that RAON is able to achieve a performance gain over Gia in terms of query success rate and query delay. We also studied the packet-level details, which gave us a very good understanding on the behavior of MANET P2P systems.

Chapter 6 Conclusion and Future Work

The research in ad-hoc networking has been mainly focused on routing and Quality of Service delivery. Lately transport and application support issues have started appearing in the research agenda. The research on application support is extremely important as the industry has started showing signs of interest in this technology for 4G wireless system [37]. Lack of a good abstraction for application support necessitates dealing with similar issues for each application design. In the Internet, overlay networking approach have generated promising results in facilitating deployment of various applications, such as P2P, multicast and to some extent VoIP. Our first major contribution in this thesis is that we propose overlay networking as a good abstraction for application design and deployment on ad-hoc networks. The principal benefit of this approach is that the application states are only maintained by the nodes involved in the application execution and all other nodes only perform networking related functions. This is a significant gain for ad-hoc networks where nodes join and participate in the network routing for an unpredictable and transient time, and maintaining application states in those nodes will obviously accrue huge cost in the application execution.

In this thesis we focused on P2P overlay network design on ad-hoc network. We base our design on Gia, which is an unstructured P2P system giving superior performance over other unstructured systems due to its flow control and biased random walk forwarding schemes. We extended the design of Gia to address problems specific to MANET. Our system, called RAON (Resource-Aware Overlay Network), makes forwarding decisions taking into consideration of the MANET characteristics of link instability and power constraints. We added two new features to Gia's basic biased

random walk forwarding: NCS (Neighbor Coloring Scheme) and PNR (Proactive Neighbor Replacement). The NCS uses a ranking system that ranks neighbors according to their available resources. It monitors the condition of the overlay links by sending update messages and measuring the delay periodically. It colors the links either green, yellow or red depending upon the link latency and the neighbor's residual power. We also proposed a supplementary algorithm, called PNR, which attempts to find a better node to replace a neighbor if the connection to the neighbor is found to be unstable. We evaluated the performance of RAON and Gia using NS2, which is a packet-level simulator. Simulation results show that RAON is able to achieve more stable and improved query success rate and query delay as compared to Gia under a variety of topology and load conditions. It, however, achieves this at the expense of higher energy consumption. The overhead of control messages during a new neighbor search in PNR is the major source of higher energy consumption. This can be controlled through more prudent use of PNR. We summarize our results with pointers to future work in the rest of this section.

For query success rate, RAON manages to sustain a relatively stable performance, with an improvement of 20-25% in the most extreme scenarios with high node speed and low density. RAON is also able to achieve lower query delay than Gia. The above performance gains are due to NCS, which avoids forwarding queries through unstable links. PNR also helps to improve the performance in both query success rate and query delay, but it is achieved at a higher cost. Currently, PNR makes its neighbor replacement decisions based on information collected at the application level only, which is not sufficient.

There are also few general conclusions that we can draw from our work, which provide an insight into the system design. First, we notice that the search performance is more dependent on underlying node density than speed considering realistic speed values we used in our simulations. Second, we realize that the approach of using reverse route to forward a query hit response back to the query originator might not be a good choice in MANET. Our simulation results indicate that the query delay in RAON is mainly caused by unstable links along the reverse path.

Third, flow control implemented at the application level is not sufficient to avoid congesting a node. Instead, flow control may need to be implemented at a lower level in order to achieve true flow control at all levels. True flow control can also help reduce energy consumption of a node by limiting the number of flows.

We also observed that TCP is more restrictive in making response to packet loss transparent to the application. Instead if the application is engaged in formulating the response to packet loss it may take different actions depending upon the type of message. Furthermore, out-of-order packet is not a real issue for P2P systems, since messages can be processed despite of their order of arrival. We suggested a hybrid approach with UDP that may resolve the problem of using unstable reverse route that leads to high query delay. However, application developer must implement a reliable version of UDP in order to detect packet loss.

Further study needs to be taken in order to find out whether UDP is a better approach than TCP for MANET P2P systems or not. It may also be helpful if the application can gain access to the underlying routing information like physical hop counts, such that it can make better decisions (forwarding, connecting, dropping, etc.) based on

more accurate information. An underlying flow control system might be out the scope of P2P design, but it is a feature that may benefit many other applications. Finally, file sharing is only one of the applications that use the P2P technology. Other applications may also exploit the results and observations that we presented in this thesis in their design process.

References

[1] Yang-hua Chu, Sanjay G. Rao, Srinivasan Seshan and Hui Zhang, "A Case for End System Multicast", in *IEEE Journal on Selected Areas in Communication (JSAC), Special Issue on Networking Support for Multicast*, 2002.

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

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

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

[5] D. Bertsekas and R. Gallager, "Data Networks", pages 297-333, Prentice-Hall, Inc., 1987.

[6] C. Hedrick,"RFC 1058: Routing Information Protocol", June 1988.

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

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

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

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

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

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

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

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

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

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

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

[18] "DirectConnect". http://www.neo-modus.com.

[19] S. Sen, and J. Wang, "Analyzing Peer-to-Peer Traffic Across Large Networks", in

Proceedings of the ACM SIGCOMM Internet Measurement Workshop 2002, November 2002.

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

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

[22] M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron, "SCRIBE: A large-scale and decentralised application-level multicast infrastructure", in *IEEE Journal on Selected Areas in Communication (JSAC), Vol. 20, No. 8*, October 2002.

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

[24] "Skype". http://www.skype.com.

[25] S. Ethier, "Application-Driven Power Management: A framework for achieving fine-grained control over the power consumption of purpose-specific mobile devices", *QNX Software Systems Inc.*

[26] R. Kravets and P. Krishnan, "Application-driven power management for mobile communication", in *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 263-277, 2000.

[27] J. Flinn and M. Satyanarayanan, "Energy-aware adaptation for mobile applications", in *Proceedings of Symposium on Operating Systems Principles*, December 1999.

[28] P. Reynolds, and A. Vahdat, "Efficient Peer-to-Peer Keyword Searching", *Technical report, Duke University, Durham, NC*, 2002

[29] Q. Lv, S. Ratnasamy, and S. Shenker, "Can Heterogeneity Make Gnutella Scalable?", in *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*. MIT Faculty Club, Cambridge, MA, USA, March 2002.

[30] Q. Lv, P. Cao, E. Cohne, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks", in *Proceedings of 16th ACM International Conference on Supercomputing (ICS'02)*, November 2001.

[31] Gnutella Development Forum, "The Gnutella v0.6 protocol, 2001", http://groups.yahoo.com/group/the_gdf/files/.

[32] D. Tsoumakos and N. Roussopoulos, "Analysis and Comparison of P2P Search Methods", *University of Maryland, Dept. of Computer Science Technical Report (CS-TR-4539)*, November 3, 2003.

[33] Z. Ge, D. R. Fiegueiredo, S. Jaiswal, J. Kurose, and D. Towsley, "Modeling peer-to-peer file sharing systems", in *IEEE Infocom 2003*, April 2003.

[34] Q. He, M. Ammar, G. Riley, H. Raj, R. Fujimoto, "Mapping Peer Behavior to Packet-level Details: A Framework for Packet-level Simulation of Peer-to-Peer Systems", in *MASCOTS 2003*, October 2003.

[35] "NS2". http://www.isi.edu/nsnam/ns/.

[36] Sung-Ju Lee, Julian Hsu, Russell Hayashida, Mario Gerla, and Rajive Bagrodia, "Selecting a Routing Strategy for Your Ad Hoc Network", *Computer Communications*, special issue on Advances in Computer Communications and Networks: Algorithms and Applications, Vol. 26, Issue 7, pages723-733, May 2003.

[37] Petri Mähönen and Janne Riihijärvi, "Co-operative and Ad Hoc Networks", Wireless World Research Forum (WWRF) Working Group 3, Version 4, April 2004. http://www.wireless-world-research.org/general_info/Documents/Charters/WG3-Charter-13Oct04.pdf.

Appendix A

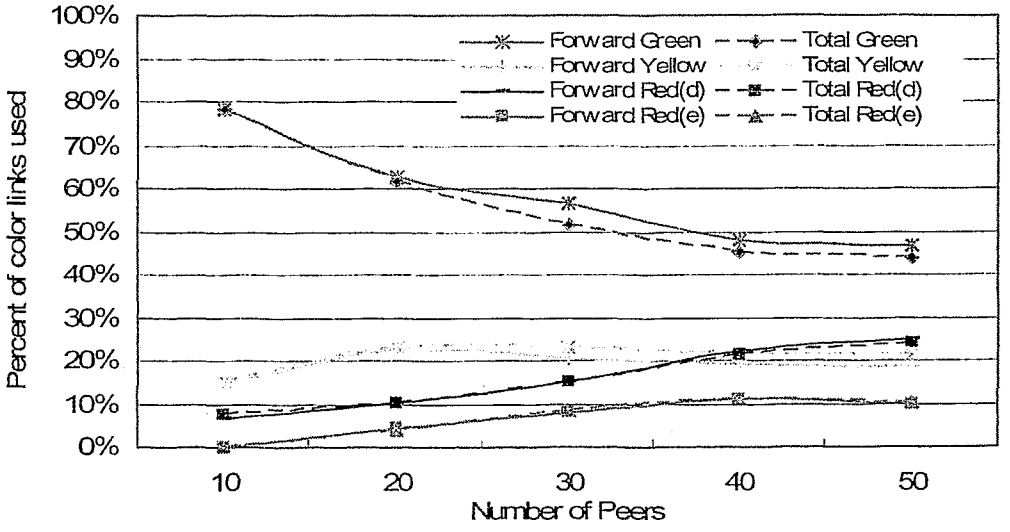


Figure A.1: Percent difference in colored links used by Gia

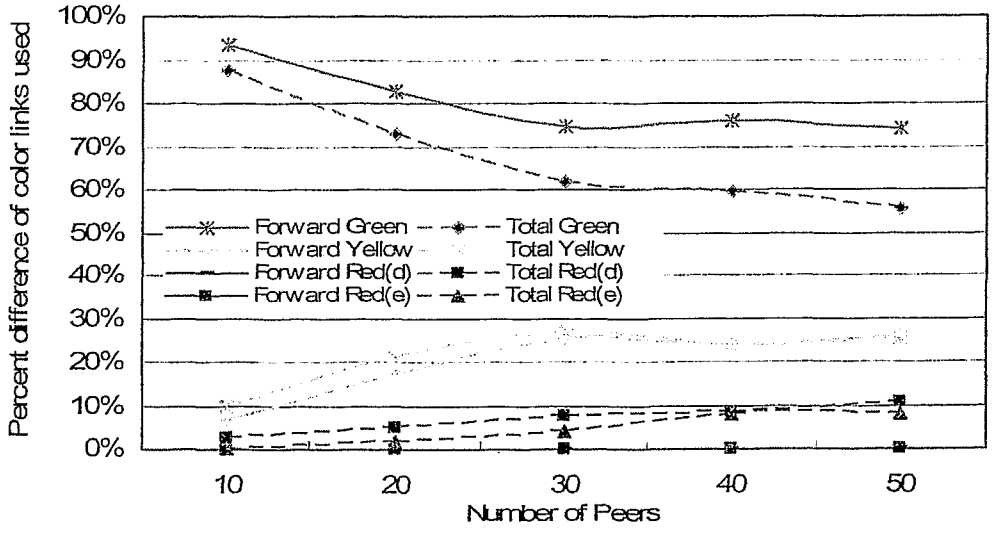


Figure A.2: Percent difference in colored links used by (RAON – PNR)

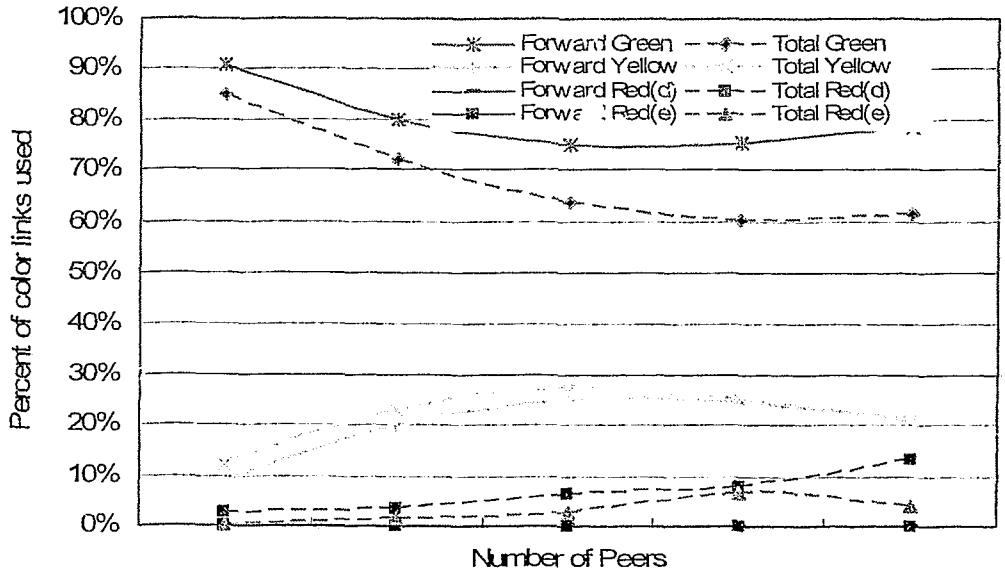


Figure A.3: Percent difference in colored links used by RAON