A PEER-TO-PEER FILE SHARING SYSTEM FOR WIRELESS AD-HOC NETWORKS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Hasan Sözer

August, 2004

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. İbrahim Körpeoğlu (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Özgür Ulusoy

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Uğur Güdükbay

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

A PEER-TO-PEER FILE SHARING SYSTEM FOR WIRELESS AD-HOC NETWORKS

Hasan Sözer
M.S. in Computer Engineering
Supervisor: Assist. Prof. Dr. İbrahim Körpeoğlu
August, 2004

In recent years, we have witnessed an increasing popularity of peer-to-peer (P2P) networks. Especially, file sharing applications aroused considerable interest of the Internet users and currently there exist several peer-to-peer file sharing systems that are functional on the Internet.

In the mean time, recent developments in mobile devices and wireless communication technologies enabled personal digital assistants (PDA) to form ad-hoc networks in an easy and automated way. However, file sharing in wireless ad-hoc networks imposes many challenges that make conventional peer-to-peer systems operating on wire-line networks (i.e. Internet) inapplicable for this case. Information and workload distribution as well as routing are major problems for members of a wireless ad-hoc network, which are only aware of peers that are within their communication range.

In this thesis, we propose a system that solves peer-to-peer file-sharing problem for wireless ad-hoc networks. Our system works according to principles of peer-to-peer systems, without requiring a central server, and distributes information regarding the location of shared files among members of the network. By means of constructing a distributed hash table (DHT) and forming a tree shaped overlay network based on the topology of the network itself, the system is able to answer location queries, and also discover and maintain routing information that is used to transfer files from a source-peer to another peer.

Keywords: Wireless Ad-Hoc Networks, File Sharing, Peer-to-Peer Networks.

ÖZET

KABLOSUZ TASARSIZ AĞLAR İÇİN BİR EŞLER ARASI DOSYA PAYLAŞIM SİSTEMİ

Hasan Sözer
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Yrd. Doç. Dr. İbrahim Körpeoğlu
Ağustos, 2004

Son yıllarda, eşler arası (P2P) ağların artan popülaritesine tanık olunmuştur. Özellikle dosya paylaşım uygulamaları, Internet kullanıcılarının büyük ilgisini çekmiştir ve şu anda Internet üzerinde fonksiyonel olan birkaç eşler arası dosya paylaşım sistemi bulunmaktadır.

Aynı zamanda, seyyar cihazlar ve kablosuz haberleşme teknolojilerindeki son gelişmeler, kişisel dijital asistanların (PDA) kolay ve otomatik bir şekilde tasarsız ağlar oluşturmalarına olanak sağlamıştır. Fakat, kablosuz tasarsız ağların ortaya koyduğu zorluklar, Internet gibi kablolu ağlar üzerinde çalışan eşler arası sistemlerin bu ortama uygulanmasını olanaksız kılmaktadır. Bilgi ve iş dağılımı ile birlikte yol atama, sadece haberleşme menzili içine düşen eşlerden haberdar olan kablosuz tasarsız ağ üyeleri için önemli problemlerdir.

Bu tez çalışmasında, kablosuz tasarsız ağlar üzerinde eşler arası dosya paylaşım problemini çözen bir sistem önerilmektedir. Önerilen sistem, merkezi bir sunucuya ihtiyaç duymaksızın, eşler arası sistemlerin prensiplerine göre çalışmakta ve paylaşılan dosyaların konum bilgilerini ağın üyeleri arasında dağıtmaktadır. Dağıtık bir kıyım tablosu (DHT) ve ağın yapısına dayanan ağaç şeklinde bir yerpaylaşan ağ oluşturmak suretiyle, sistem hem konum sorgularını cevaplandırabilmekte, hem de dosyaların kaynak eşten diğer eşlere aktarımında kullanılan yol atama bilgisini keşfedip bu bilgiyi güncel tutabilmektedir.

Anahtar sözcükler: Kablosuz Tasarsız Ağlar, Dosya Paylaşımı, Eşler Arası Ağlar.

Acknowledgement

I would like to express my gratitude to my supervisor Assist. Prof. Dr. İbrahim Körpeoğlu for his instructive comments in the supervision of the thesis.

I would like to express my thanks and gratitude to Prof. Dr. Özgür Ulusoy and Assist. Prof. Dr. Uğur Güdükbay for evaluating my thesis.

I would like to express my thanks to Metin Tekkalmaz for his accompany during the research.

I would like to express my thanks to Burcu Kaplanlıoğlu, Burcu Ayşen Ürgen and Selen Pehlivan for the implementation of the simulation environment.

I would like to express my thanks to Hüseyin Özgür Tan and Tağmaç Topal for their valuable comments about this work.

I would like to express my thanks to ASELSAN A.Ş. for permitting and supporting me to continue my M.S. education and research along with my occupational commitments.

I would like to express my thanks to The Scientific and Technical Research Council of Turkey (TÜBİTAK) for supporting this work (Grant Number: 103E014).

I would like to express my special thanks to my parents for their endless love and support throughout my life...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Peer-to-Peer (P2P) networks have been very popular since their first emergence. Especially, file sharing applications aroused considerable interest of the Internet users and several systems have already been deployed to be functional, like Napster [10], Gnutella [4] and Fasttrack [3]. These systems currently serve many users who are able to share files located at their PCs. Together with the new users of the Internet and the emergence of different types of files to be shared (documents, audio files, etc.), number of users of peer-to-peer file sharing systems increases every day.

In the mean time, mobile devices and wireless communication technologies are evolving and becoming very popular. Both areas have experienced rapid improvements during last few years, which led to development of high-performance products. Today, personal digital assistants (PDA) have almost the same abilities that of ordinary PCs despite their small size and weight. On the other hand, new wireless technologies enable PDAs and other handheld devices to communicate and form ad-hoc networks in an easy and automated way. Bluetooth [2], for instance, is such a technology that uses short-range radio communication and that interconnects handheld electronic devices ranging from cellular phones to PDAs.

Although high-performance handheld devices that communicate with each other through ad-hoc wireless communication technologies are available today, peer-to-peer file sharing in such an environment imposes many challenges that make conventional peer-to-peer systems operating on wire-line networks inapplicable for this case. Essentially, peer-to-peer systems were developed as opposed to central approaches in order to increase availability and reliability. They eliminate the existence of a single point of failure and facilitate load balancing. In that respect, they are very suitable for wireless ad-hoc networks (WANETs) where spontaneous connections occur and users have relatively higher degree of mobility. However, traditional peer-to-peer systems are not sufficient for providing file sharing in such an environment since:

- Such networks can be formed anytime and anywhere without requiring an infrastructure,
- Nodes in the network may tend to change their locations more frequently,
- There is lack of widely accepted and used standards for routing data in wireless ad-hoc networks.

A peer-to-peer file sharing system that is running on the Internet may find a desired file at a member node, which is identified by a unique ID. This can be achieved by using centralized or distributed indices that maps the name of the file to the member node's IP address through which the node can be reached. After knowing the IP address of the node from where a file can be downloaded, the network layer of the Internet (IP) would handle all intermediate steps and forwarding needed in order to reach to the node and to perform the download. However, this is not possible on a WANET that does not run an ad-hoc routing algorithm. A WANET may be composed of heterogeneous mobile devices in which a standard routing algorithm is not supported at all nodes. In that case, nodes of the WANET would only be able to communicate with other nodes that fall into their communication range. Currently, although there are various efforts ([13]) that propose protocols to route packets in a WANET, we still lack a common and widely used standard routing protocol for this environment. And it seems that it will take some more time before we have a widely accepted common routing protocol and its implementation available and deployed.

Therefore, to support peer-to-peer file sharing in a WANET, we believe that a peer-to-peer system should also provide routing functionality besides providing lookup functionality. In this way, the peer-to-peer system should be able to determine both from where and how to obtain a file.

Peer-to-peer file sharing systems which have been proposed up to now mostly presume that an underlying network layer exists and they only provide lookup functionality. The majority of them were designed to be operated on the Internet. In other approaches, file searches are performed by flooding the network, which would lead to congestion in result of large number of submitted queries.

In this thesis, we propose a system that solves peer-to-peer file sharing problem in wireless ad-hoc networks. Our system works in a peer-to-peer manner and distributes information regarding the location of files among members of the network. Along with the location information, the system also stores routing information. While designing the system, we have adapted some techniques from source routing and peer-to-peer location lookup methods that were previously proposed for wire-line networks.

We performed experiments on a simulation environment for measuring the traffic overhead of our system, which is defined as the number of messages exchanged among members of the network. The simulation results showed that our system enables efficient access to shared files and also it is scalable though, frequent disconnections would lead to a high traffic overhead.

The remainder of this thesis is organized as follows. In the next chapter, related previous studies are summarized. In Chapter 3, an overview of the system is given, which is followed in Chapter 4 by a detailed description of each operation supported by the system. In Chapter 5, we present working scenarios of the system to show how these operations work and collaborate in order to update and maintain the distributed location and routing information. Chapter 6 introduces the simulation environment developed for measuring the overhead of basic operations of the system and in the same chapter, results we obtained are presented and evaluated. Finally, in Chapter 7 we give our conclusions and discuss some future work issues.

Chapter 2

Related Work

Among various systems designed and proposed, one of the earliest and most popular file sharing system is Napster [10], which enables file sharing among PCs connected to the Internet. It uses a central server on which location information (filename and address pairs) regarding all shared files is stored. All queries are forwarded to this server and once the location of a file is determined, file transfers are carried on in a peer-to-peer manner. Although location lookup is easier by using a central server, this system is subject to typical weaknesses of centralized systems like low reliability and availability.

More recent works concerning file sharing on the Internet aim fully distributed peer-to-peer systems that store location information in a distributed manner. CAN (Content-Addressable Network) [12], being one of them, is based on a distributed hash table (DHT). In CAN, filenames are hashed and mapped to points on a d-dimensional space. The d-dimensional space is divided into chunks and distributed among the members of the network where each member is responsible from one portion of the space (i.e. a chunk). Every member stores location information of files that are mapped to a point inside its chunk. Distribution of chunks represents an overlay network in which, members that are responsible for adjacent chunks are connected to each other. Location information of a file is reached by routing queries on this overlay network. Each member forwards the query to one another that is responsible for the nearest chunk to the point representing the searched file, until the chunk containing the point is reached. Chord [14] is another well known fully distributed peer-to-peer system using a DHT in which a ring shaped overlay network is applied. Each member on this ring stores the location information regarding a fraction of shared files and it maintains pointers to other members at various distances. To gather the location information of a file, these pointers are followed to forward queries so that the access path is as short as possible. Although these systems work in a fully distributed manner, they cannot be applied to WANETs unless there is a support for routing functionality and addressing mechanism like IP. This is because; both systems are using overlay networks, which does not reflect the physical network. Adjacent nodes in these overlay networks may actually happen to be many hops of distance away from each other. Tapestry [15], as another peer-to-peer system based on a DHT, makes use of a topology aware overlay network. The approach used in this system guarantees that latency caused by routing of a message on the overlay network is proportional to the latency that would be faced in directly routing of it on the underlying physical network. While this approach reduces the overhead of overlay routing, still the system considers only routing on the overlay network where single overlay-hop may correspond to multiple hops on the underlying network.

Another system proposed recently in [9] and works in a peer-to-peer manner is based on probabilistic flooding. In this approach, each member forwards a query to all of its neighbors with some probability. If the destination is reached, which is not guaranteed, location information is cached. Next time the same query is received, it is forwarded directly to the destination by using the information stored in the cache. Again, this approach was proposed for the Internet. That is why, how multi-hop communication can be achieved is not considered.

There exist several other peer-to-peer file sharing systems ([1]) designed for the Internet, yet for that reason, they do not consider the routing of information. As the case with systems that are mentioned, they make use of the IP protocol of the Internet.

First work about file sharing on WANETs is 7DS [11], which enables members of the system to browse the web with an intermittent Internet connection. Whenever a node fails to connect to the Internet, thus cannot reach a web page, it searches for the required data among peers. If the data being searched can be found in one of the peer's cache, it is transferred to the source of the query. This system though, assumes that there exist Wireless LANs, through which nodes can communicate with each other. Besides, nodes can only communicate with other nodes in close proximity. In other words, in this system, multi-hop communication is not possible between nodes that are not in the same wireless coverage.

Although most of the previous studies focus only on lookup functionality without concerning about how routing of queries will be handled, independently, there have been various efforts ([13]) that propose protocols to route packets on WANETs. One of the popular ones is DSR (Dynamic Source Routing) [5], which is based on source routing and flooding. Each node in the network forwards packets to all of its neighbors unless it is the destination. The path that is traversed by the packet is recorded in it along the way, so that, packets sent between source and destination can be routed by means of this information thereafter.

As a matter of fact, flooding together with source routing seems to be an adequate solution for file sharing problem in WANETs. Queries can be flooded on the network. In result, a query packet would reach to the owner of the shared file. After that, file transfer can take place through the route recorded in this packet. Such a system does not require any infrastructure, routing functionality, or distribution and maintenance of some sort of location information. This fact is realized by some studies ([6, 8]), which propose peer-to-peer file sharing systems for WANETs based on flooding. Regardless of its advantages however, flooding leads to a traffic overhead. Each query is forwarded to all nodes in the network. In result of large number of queries submitted, the network would inevitably be congested. In [6] and [8], mechanisms like caching and selective routing are proposed in order to prevent flooding of the entire network. Such approaches work fine for small-size WANETs but as the network gets bigger, they cause traffic overhead and the probability of finding a file in the network reduces.

Our system forwards unicast location queries and provides a deterministic way to locate and access files. Hence if a file is shared in the WANET, its location can be determined and it can be accessed. Also, as a difference from previous studies, we focus on and provide solution for both lookup functionality of a file sharing system and multi-hop routing issues on WANETs. We propose a fully distributed, peer-to-peer, cross-layer system, which merges lookup functionality of the application layer and routing functionality of the network layer.

Chapter 3

System Overview

The system expects three basic functionalities listed below from the underlying network layers, which are basically the physical layer and link layer functionalities.

- Device discovery
- Communication with nodes in the range
- Notification of link failure

In order to form a file sharing enabled WANET, each node should be aware of other nodes that are within its communication range. This awareness is supported by the device discovery functionality. As another functionality to be carried on by the underlying network layers, node pairs that are aware of each other should be able to establish a connection and they should be able to communicate (i.e. send/receive messages to/from each other). In result of mobility or due to problems in the wireless channel, a formerly established connection may be lost after which exchanging messages between two ends of the connection is no more possible. In such cases, the underlying network layers of corresponding nodes should notify upper layers (i.e. our system) about the link failure.

Along with these functionalities, the system makes use of a fully distributed hash table where keys are names of shared files and values are globally unique locations of these files (MAC address of the device together with the full path of the file on the device may provide this uniqueness) together with necessary routing information which will be described soon. The basic dynamics of the system is as follows. A one-dimensional space (i.e. a line) is used to store (key, value) pairs by mapping each key to a point P on this space, namely the *hashline*, using a uniform hash function (See Figure 3.1). In fact, any hash function that can map a file name to a real number between 0 and 1 may be used for this purpose. However, uniformity would lead to a more balanced information distribution among the nodes. Each node in the WANET is responsible for storing a segment of the *hashline* (i.e. hash table entries which correspond to points that are included in this *hashline* segment).

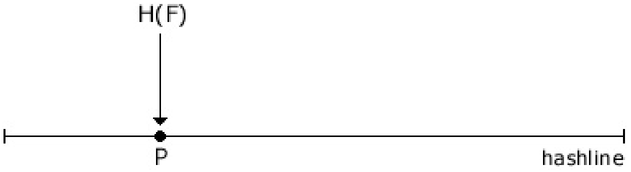


Figure 3.1: A file is mapped to a point on the *hashline*, where the file name is F and the corresponding point is P.

We call the node, which is responsible for the segment of *hashline* containing a point P as *P-Node*, and the node, which stores a file with name F as *F-Node*. Hence, a *P-Node* stores index information (i.e. F) along with corresponding location and routing information, and an *F-Node* stores the actual file.

At the highest abstraction level, a file is accessed by following the steps listed below.

1. Name of the file to be searched is hashed to determine a point P on the *hashline*.
2. *P-Node* is accessed.
3. The location of the searched file, *F-node*, and the route to that location is determined from *P-Node*.
4. *F-Node* is accessed, and the file is downloaded.

These steps seem simple but determining routes between nodes is the heart and distinguishing part of the system. System is designed to cope with this problem using a tree shaped overlay network that is imposed on the nodes of a WANET (See Figure 3.2). The overlay network helps in accessing to *P-Node*, and the information obtained from *P-Node* helps in determining the route to *F-Node* from where the file will be downloaded.

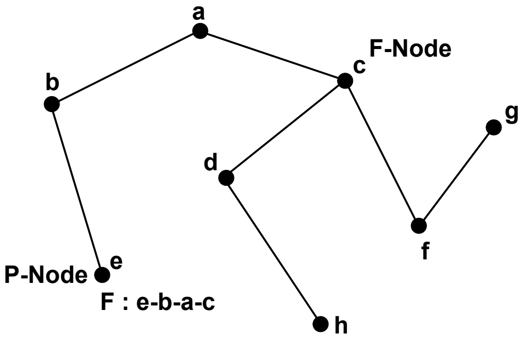


Figure 3.2: A tree shaped overlay network is applied for routing.

Unlike overlay networks used in peer-to-peer systems previously proposed, the overlay network used by our system is based on the physical connectivity of nodes. In other words, an overlay-hop corresponds to a single hop in the WANET. In

Figure 3.2, for instance, nodes a and b are connected in the overlay network. That means, these nodes are in the communication range of each other.

As another property of the overlay network used by the system, loops do not exist although the actual network may include loops at the link layer. By definition, the "tree" shaped overlay network is cycle-free. While the network grows with the addition of new members, a new member node is not permitted to join the same file sharing enabled WANET via more than one link (i.e. via more than one neighboring node). A loop-free network can be achieved by providing a unique network ID (e.g. MAC address of the root node) for each file sharing enabled WANET and not allowing a node to have more than one parent with the same network ID. Considering the overlay network in Figure 3.2 again, nodes e and d may be in the communication range of each other. However, even so, they would not connect to each other because they have the same network ID, which is the MAC address of node a. When a new node connects to the system, it obtains the Network ID from the node to which it is connected (i.e. the parent).

Chapter 4

Operations Supported by the System

There are several operations supported by the system to locate files and route the download for enabling file sharing. The *Node-Join* operation is executed when a node is connected to a file sharing enabled WANET. It might be the case that two file sharing WANETs are formed independently. When they merge, the *Network-Join* operation is executed. The *Access2P-Node* operation is used to find and access the node which stores segment of the *hashline* including a desired point P. The *Access2F-Node* operation is used to find and access the node which stores a desired file with name F. The *Insert* and *Delete* operations are used to add a file to the network (i.e. enable sharing) or remove a file from the network, respectively. The *Recover* operation is executed in order to preserve the consistency between the actual location of shared files and the hash table storing the routing information when a disconnection with an adjacent node is detected. Finally, the *Leave* operation is executed when a node decides to leave the file sharing enabled WANET.

Essentially, operations mentioned above are highly dependent on each other. Some of them include other operations (e.g. The *Node-Join* operation includes *Access2P-Node* and *Insert* operations). Following subsections contain detailed information about each operation. Subsequent chapter, on the other hand, goes over sample scenarios in order to enlighten the big picture and to illustrate cases in which these operations work and collaborate.

4.1 Node-Join

Whenever a node **N** decides to join a file sharing enabled WANET, the following steps are executed:

1. **N** connects to an already existent node **K** of the network, which is accomplished by the underlying protocols specific to the WANET (See Figure 4.1(a)).
2. **K** assigns half of its segment of *hashline* to **N** and passes the related hash table entries to it (See Figure 4.1(b)).
3. **N** adds **K** to the routing path information maintained at each hash table entry for files indexed at **N** before saving the hash table entries (See Figure 4.1(c)).
4. **N** assigns **K** as its parent and **K** adds **N** to its children list in the logical tree structure (i.e. overlay network).
5. **N** calls the *Insert* operation for each file it wants to share and whose hashed value is out of its responsibility.

As it can be noticed, the *hashline* segment assigned to the new node is not randomly determined, which is the case in [12]. Instead, the node, to which the new node directly connects, shares half of its responsibility on the *hashline*. This simple design is crucial for easy and efficient routing of location queries to the nodes that can answer them, which is explained together with the discussion on the *Access2P-Node* operation.

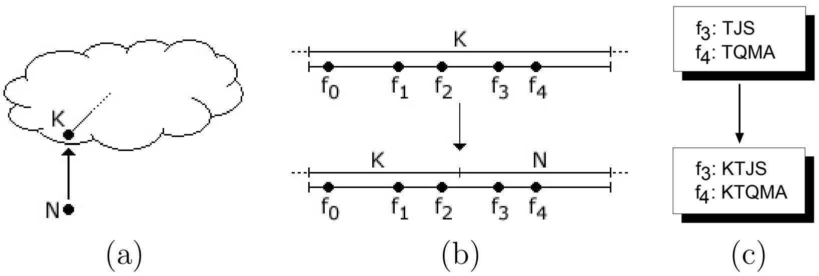


Figure 4.1: Basic steps of the *Node-Join* operation; (a) Establishing the connection, (b) Sharing the *hashline* segment, and (c) Passing the hash table entries.

4.2 Network-Join

Let nodes **N** and **K** be the members of two distinct file sharing enabled WANETs, referred as **N-Net** and **K-Net** respectively, which are going to merge by **N**-**K** connection. To obtain a larger file sharing enabled WANET from two smaller ones, the following steps are executed, assuming that **K** first discovers **N** and connects to it:

1. Every node of **K-Net** on the path from **K** to the root of **K-Net** (node with no parent), exchanges the parent-child role with its parent, including **K** and the root of **K-Net**. That is, every node on the specified path adds its former parent to its children list and it becomes the parent of its former parent. In this way, **K** becomes the new root of **K-Net** (See Figure 4.2(b)).
2. **K** is connected to **N**. Hence, **N** becomes the parent of **K** (See Figure 4.2(c)).
3. Based on new parent-child relationships among **K-Net** nodes and **N**, starting from **N**, each parent shares half of its responsibility on the *hashline* with its children, in an iterative manner.
4. Each node in **K-Net** calls the *Insert* operation for each file it wants to share.

As a result of the *Network-Join* operation, the *hashline* previously maintained in **K-Net** becomes invalid and it is discarded. Members of **K-Net** retake their

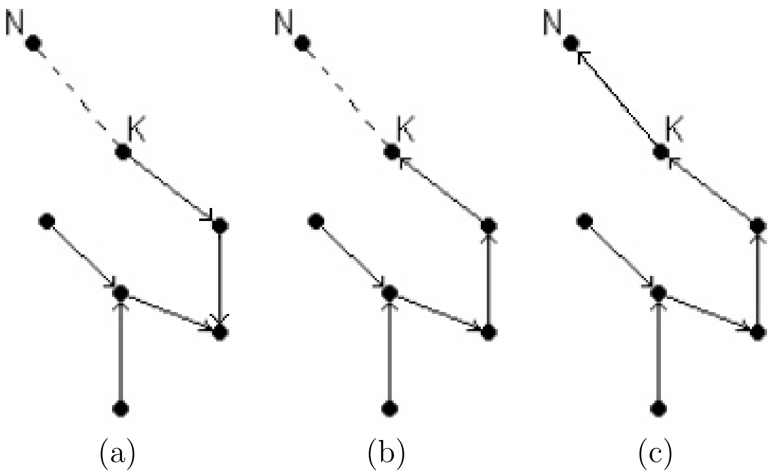


Figure 4.2: Basic steps of the *Network-Join* operation; (a) Discovering the member of another network, (b) Exchanging parent-child roles, and (c) Establishing the connection.

hashline segments, which are sub segments of the *hashline* segment that is owned by **N**, and they become a member of **N-Net** thereafter.

4.3 Access2P-Node

Whenever a node **N** wants to access *P-Node* (i.e. the node which is responsible for the segment of the *hashline* containing point P), it invokes the *Access2P-Node* operation. A node **K** receiving the *Access2P-Node* request follows these rules:

1. If point P is included by the segment of *hashline* that **K** is responsible for: *P-Node* is found and is **K**.

2. If point P is included by the segment of *hashline* that one of the children of **K** is responsible for: **K** adds itself to the route list and forwards the *Access2P-Node* request to the relevant child node.

3. Otherwise: **K** adds itself to the route list and forwards the *Access2P-Node* request to its parent.

Note that initially **N** = **K**. Also note that *P-Node* finally has the routing information between the node issuing the *Access2P-Node* request (i.e. **N**) and itself, since each node on the path from **N** to *P-Node* adds itself to the routing path information that is carried inside the *Access2P-Node* request.

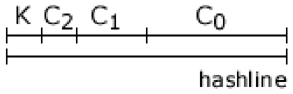


Figure 4.3: The *hashline* segment owned by a node, **K** and its children.

Suppose, **K** has three children C_0, C_1 and C_2, which share the *hashline* segment as depicted in Figure 4.3. If an *Access2P-Node* request reaches to **K**, first **K** checks whether the interested point (i.e. P) falls into its responsibility. If not, it checks the *hashline* segments for which its children are responsible. Since, **K** had previously assign these segments to its children, it has this information. Assume, C_0 is responsible for the point P. It might have been the case that many other nodes are connected to C_0 in time. So, currently point P may not be in responsibility of C_0, but one of its children. **K** does not know and concern that. It just forwards the request to C_0. C_0 will further forward the request to its children, if necessary. When any of *hashline* segments owned by **K**, C_0, C_1 and C_2 do not include the point P, **K** forwards the request to its parent.

4.4 Access2F-Node

Whenever a node **N** wants to access *F-Node* (i.e. the node which contains the file with name F), it invokes the *Access2F-Node* operation, which consists of the following steps:

1. **N** hashes F and determines P. That is, P = Hash(F).
2. Having point P, **N** invokes the *Access2P-Node* operation with *F-Node* location request. That is, **N** asks *P-Node* the route from *P-Node* to *F-Node.*

3. Having the route information back to **N**, due to the feature of the *Access2P-Node* operation, P-Node sends to **N** the route from itself to *F-Node* (Remember that the route from *P-Node* to *F-Node* is stored as a part of the hash table entry corresponding to point P).

4. **N** combines the route information from itself to *P-Node* and from *P-Node* to *F-Node* and constructs the route necessary to access *F-Node*.

5. **N** discovers cycles in the constructed route by detecting repetition of node IDs and eliminates them.

6. **N** accesses *F-Node* by means of the generated cycle-free route.

4.5 Insert

Whenever a node **N** wants to share a file with name F, it invokes the *Insert* operation, which consists of the following steps:

1. **N** hashes F and determines P. That is, P = Hash(F).

2. Having point P, **N** invokes the *Access2P-Node* request with insertion as the request type and F as the filename.

3. Upon receiving the request, P-Node stores the filename F and the route information back to N, which is stored in the *Access2P-Node* request during traversal of the route, as a part of the hash table entry created.

4.6 Delete

Whenever a node **N** wants to stop sharing a file with name F, it invokes the *Delete* operation, which consists of the following steps:

1. **N** hashes F and determines P, That is, P = Hash(F).

2. Having point P, **N** invokes the *Access2P-Node* operation with deletion as the request type and F as the filename.

3. Upon receiving the request, *P-Node* removes the entry for the file with name F from the hash table.

4.7 Recover

The *Recover* operation is executed when a disconnection is detected with one of the neighbor nodes. The node to which the connection is lost can be the parent node or a child node. These two cases are treated separately as follows.

Whenever a node **N** determines a disconnection with one of its child nodes **K**:

1. **N** regains the responsibility of the segment of *hashline* that **K** was responsible for.

2. **N** broadcasts to the WANET a message that includes information about the regained segment to force all the nodes to invoke the *Insert* operation again for the files whose hashed names are included by the segment that **K** used to be responsible for. In this way, **N** will have hash table entries created for these files.

Whenever a node **K** determines a disconnection with its parent node **N**:

1. **K** takes full *hashline* as the area of responsibility.

2. Starting from **K**, each parent shares half of its responsibility on the *hashline* with its children.

3. Each node calls the *Insert* operation for each file it wants to share.

After a node **N** determines a disconnection with one of its child nodes and regains the responsibility of the corresponding segment of the *hashline*, it is possible that **N** would be responsible for segments that are not adjacent, thereafter.

During the execution of the *Node-Join* operations, such segments are distributed first, after which the existing one is divided into two. In Figure 4.3, for instance, when C_0 disconnects, **K** will take back its segment and will be responsible for two segments. As soon as another child node connects to **K**, this segment will be given to it. When there is no such a disjoint segment available, the existing one will be shared.

4.8 Leave

When a node **N** wants to leave the file sharing enabled WANET, it invokes the *Leave* operation, which consists of the following steps:

1. **N** invokes the *Delete* operation for each file it shares after which all index information about the files stored in **N** is removed from the WANET.
2. **N** gives its responsibility on its segment of the *hashline* to its parent.
3. **N** informs its parent P_N and children C_1, C_2, $\ldots$, C_n about its departure to make sure P_N adds C_1, C_2, $\ldots$, C_n to its children list and C_1, C_2, $\ldots$, C_n assign P_N as their parent.

Note that the third step is possible only if all children nodes of **N** are in the communication range of P_N. For children nodes that are not in the communication range of P_N, the *Recover* operation is executed.

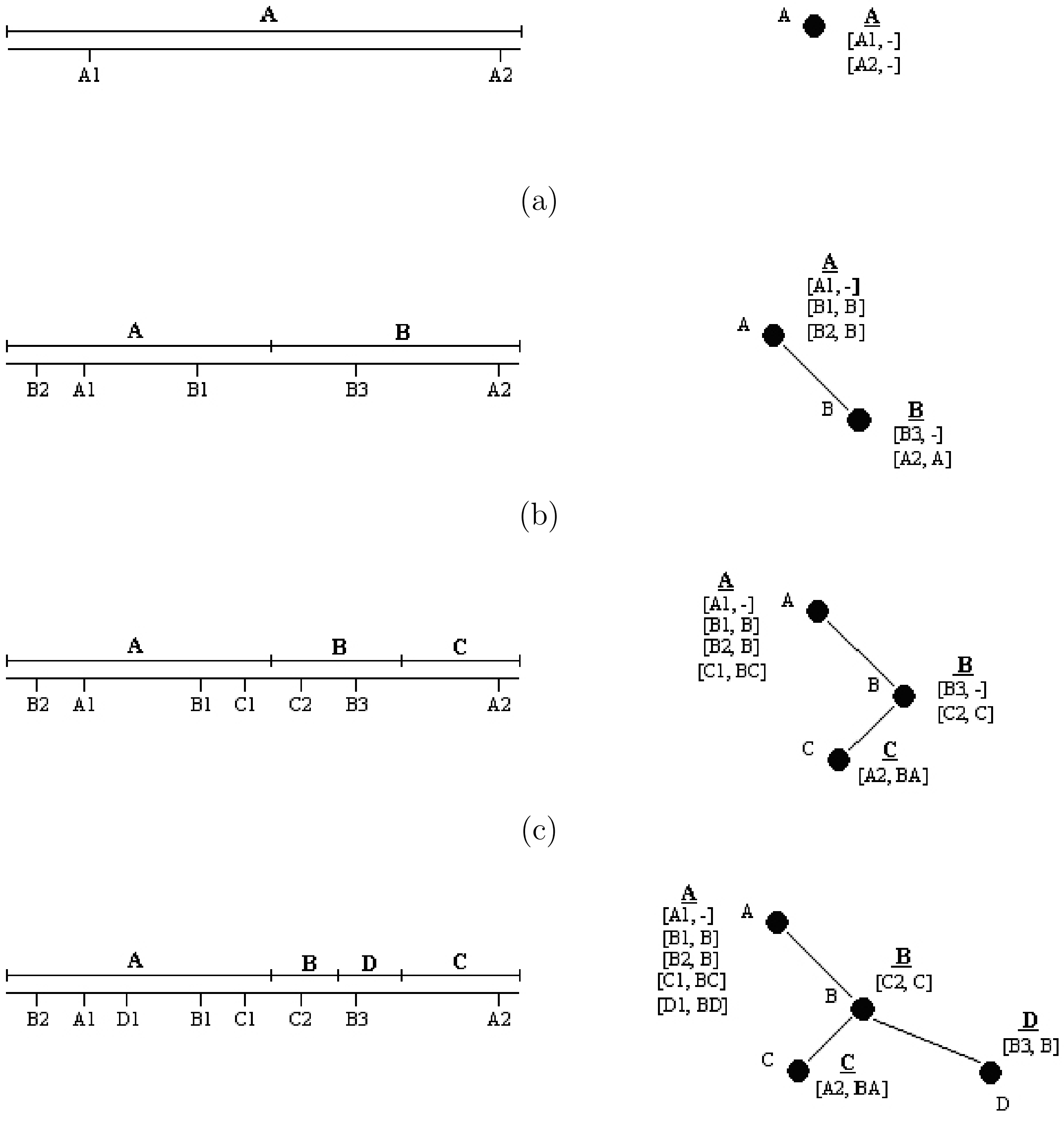
Due to the nature of ad-hoc networks, nodes are not expected to leave the network with notification. Nevertheless, it may be the case when they leave the file sharing enabled WANET on purpose, where *Leave* operation is beneficial. Otherwise, the *Recover* operation still handles the situation despite its higher traffic overhead.

Chapter 5

Sample Scenarios

After specifying each operation supported by the system, this chapter presents sample scenarios starting from the very beginning of the network formation in which the way that system works can be observed. Suppose that initially two nodes called A and B meet. A includes files A1, A2, while B has B1, B2, B3. B discovers A, in other words, B joins the network, which is only composed of A. Previously, A was responsible of all *hashline* and files A1 and A2 were mapped on to this line as depicted in Figure 5.1(a) As explained in the previous chapter, when B is connected to A, A divides the entire *hashline* into two halves and gives one of them to B. Since, A2 falls within the segment that B is now responsible for, A sends the location information for file A2 to B. Previous location information for A2 was null, meaning that the file was stored at the same node where the location information is kept. But, from now on, B stores an index entry for A2 with location information like [A2, A]. Then B executes the *Insert* operation for files B1 and B2, since these are files owned by B but they are not mapped to the part of the *hashline* that B is responsible for. Now, A stores location information, [B1, B] and [B2, B], for these files as depicted in 5.1(b).

Suppose that a new node C discovers B and connects to it. Again, the *Node-Join* operation will be invoked and the *hashline* segment that B is responsible for will be divided into two parts, as depicted in Figure 5.1(c). C stores and shares files C1 and C2, which map to points on the *hashline* as shown in the



(a)

(b)

(c)

(d)

Figure 5.1: The *hashline* state, network topology and information distribution for the network composing of (a) a single node, A (b) two nodes, A and B (c) three nodes, A, B and C (d) four nodes, A, B, C and D.

figure. First of all, B sends information about A2 to C, since A2 falls now in C's segment of responsibility. C should not only keep information about the node where file A2 can be found, but also keep route information about how it can be reached from C to that node. Therefore, C adds also B to the route information and stores an index entry as [A2, BA]. This indicates that file A2 is stored at node A (right-most node in the path) and the path from C to that node is "CBA". Next, C invokes the *Insert* operation both for C1 and C2. C1 maps to the segment controlled by A and C2 maps to the segment controlled by B. Therefore, the *Access2P-Node* request reaches to B for file C2, and to A for file C1. So, corresponding nodes store indices together with the corresponding route information to the node where related files are actually stored. The route information is obtained during the path traversals of the *Access2P-Node* requests. The current state of location and routing information that is maintained in the network can be observed in Figure 5.1(c). As the last member of the network, D discovers B and connects to it. B, again divides the segment of *hashline* it is responsible for into two parts and sends information about B3 to D. After that, D sends information about a single file it owns, D1 to A using the *Insert* operation. Final view of the *hashline* and the network topology together with distributed index and location information can be observed in Figure 5.1(d).

Now, assume that D needs file A2. D does not know where the file A2 resides or even whether such a file exists or not. However, according to the hash value of the filename, it is known that this information is held by another node. D has only one neighbor, B (as its parent) to which the query is forwarded. So, B receives the query, expressed as [A2, D], meaning that file A2 is requested by D. B has two neighbors, A and C. According to the hash value of the filename and the current state of the *hashline*, B decides to forward the query to C. This is because B knows that one of its children, C in this case, is responsible for the segment of the *hashline* that includes the point representing the hash value of the name of the requested file. Otherwise, B was going to forward the query to its parent, A. When query is forwarded to C, it is not guarantied that it will be answered by C. C may have some other nodes connected to it meanwhile, so it may further forward the query to one of its children again by looking within which segment the point lies. However, it does not matter for B whether C or some descendant of it answers the query. B only knows that query should be forwarded towards C in order to be resolved. For this particular case, C does not have any child and C holds the location information for A2. The path to source at which the query is initiated is also attached to the query. In this way, C receives a query [A2, BD], which means that node D requested file A2 and its request reached through node B. This path is used in order to send the query response (location information), [A2, BA], back to node D. C generates a query response message, [A2, BA], targeted to D and including the source route information "CBD" that gives the path to be followed. C passes the response to the next node on the path which is B. Again by looking to the route information in the response message, B passes the message to the next node on the path, which is D. D is the originator of the query to locate file A2. D receives the query response message and the message includes the location information [A2, BA]. Now, D knows that the file A2 is located at node A and D also knows two paths: the path from D to C (the node which holds the location information) and the path from C to A (the node which stores the file). D concatenates those paths and obtains "D-B-C-B-A". Then, it eliminates the unnecessary loop "B-C-B". The result is "D-B-A", the path from D to A. This is the path from query originator D to the node A that stores and shares the file A2. By means of this path, file A2 can now be directly reached and downloaded from A. These steps are depicted in Figure 5.2(a)–(c).

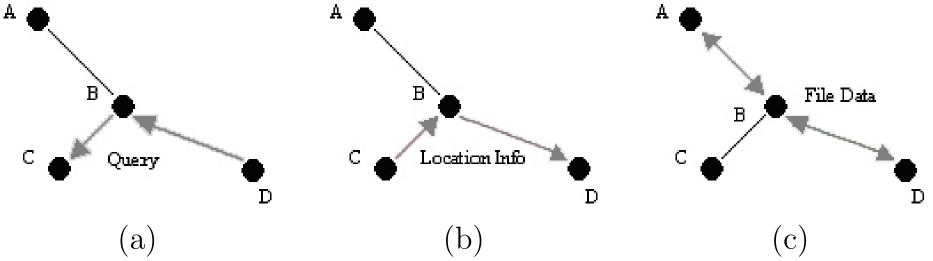
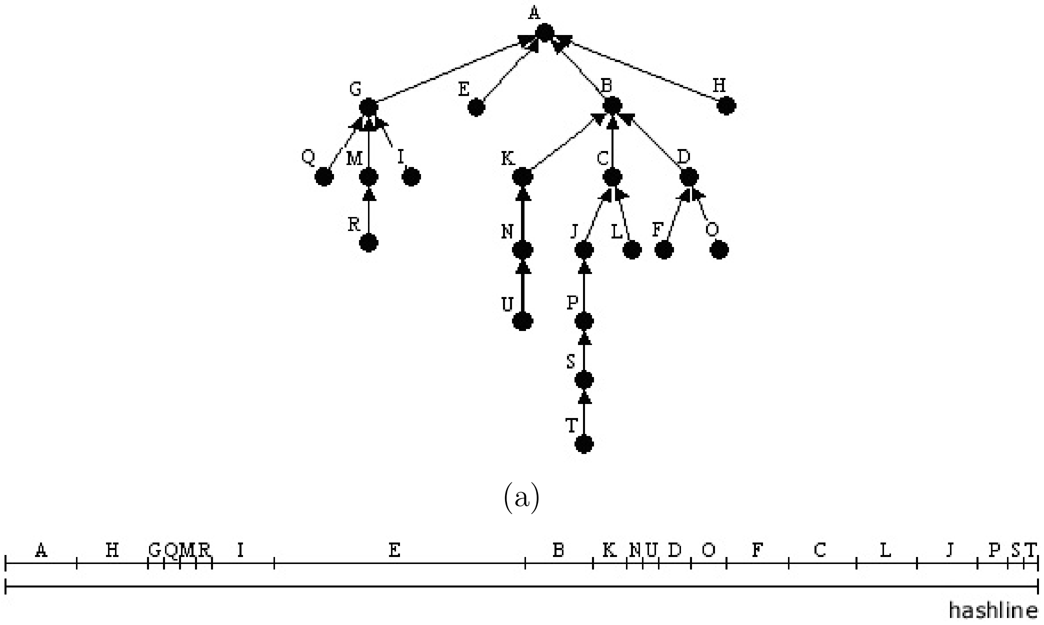


Figure 5.2: Basic steps of file search and retrieval; (a) Sending the query, (b) Receiving the location information, and (c) Acquiring the file data.

As more nodes join to the file sharing enabled WANET through the *Node-Join* operation, the tree shaped overay network becomes more involved. In Figure 5.3, a later phase of the network shown in Figure 5.1(d) is given. New nodes have joined to the WANET in alphabetical order. Connections between nodes can be inferred from the tree structure given in Figure 5.3(a). The state of the *hashline* is shown in Figure 5.3(b).



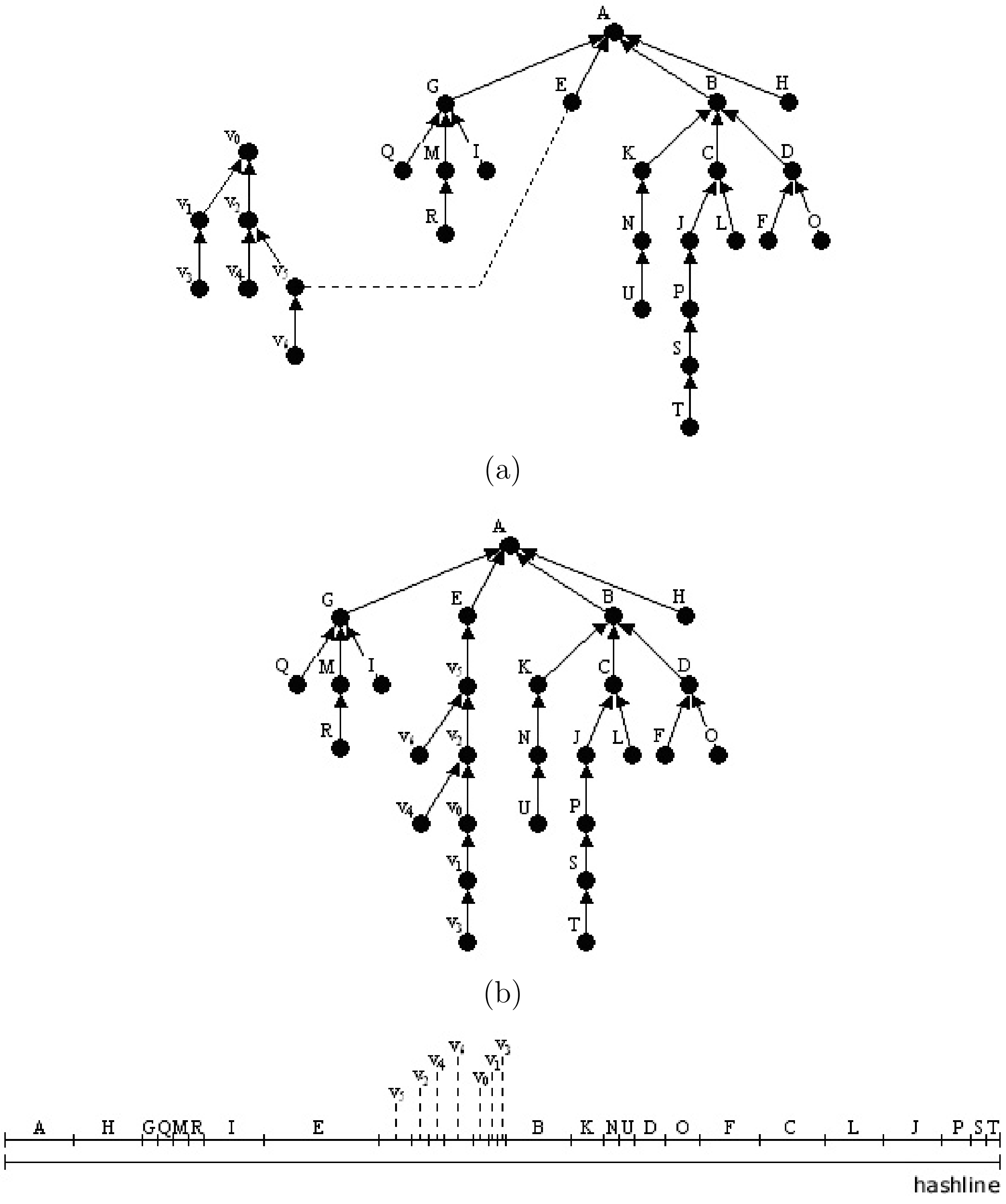
(a)

(b)

Figure 5.3: (a) A file sharing enabled WANET with 21 members, and (b) the corresponding *hashline* segmentation.

Now, consider the case where the file sharing enabled WANET in Figure 5.3 (WANET-1) merges with another file sharing enabled WANET shown in Figure 5.4(a) (WANET-2) and assume that the connecting nodes are E of WANET-1 and v_5 of WANET-2 as depicted in the figure. For such a merge operation, the *Network-Join* operation, which is explained in the previous chapter, is executed where nodes **N** and **K** in the operation correspond to the nodes E and v_5 in this sample scenario, respectively. Due to the Step 2 of the *Network-Join* operation all nodes on the path from node v_5 to the root node v_0, (i.e. v_5, v_2, v_0) exchange their parent-child relationships. The resulting parent-child relationships are depicted in the subtree, rooted at v_5, of the combined network shown in Figure 5.4(b).

Once the subtree rooted at v_5 is built, E shares a portion of its responsibility on the *hashline* with v_5. All descendants of v_5 share their responsibility on *hashline* in a similar manner, iteratively. One possible distribution of responsibilities on the *hashline* among the nodes of the new combined network is depicted in Figure 5.4(c). It should be noted that the resulting distribution may differ due to the order of children that a parent shares its responsibility with.



(a)

(b)

(c)

Figure 5.4: Unification of two file sharing enabled WANETs by means of the *Network-Join* operation; (a) Two WANETs before unification, (b) The unified WANET, and (c) The corresponding *hashline* state.

Chapter 6

Simulation and Results

In order to measure the traffic overhead of operations of the system on the network, we designed a simulation application. We define the number of messages exchanged among nodes of the network from initiation to the completion of an operation, as the traffic overhead of that operation. In the first section of this chapter, simulation environment is presented and in the second section, simulation results are discussed.

6.1 Simulation Environment

Besides what an operation does, the number of messages that would be exchanged heavily depends on the number of nodes, the structure of the topology and how close nodes involved in the operation are to each other. For instance, if a node initiates the *Access2P-Node* operation and if the corresponding *P-Node* happens to be itself, there will be no messages exchanged. Conversely, the topology can be structured as a chain of nodes where initiator and target of the *Access2P-Node* operation lies at two ends of the chain. In that case, which is the worst case, $N - 1$ messages will be exchanged, where N denotes the number of nodes.

In our simulation environment, N is one of the simulation parameters. We performed measurements regarding the traffic overhead of operations in varying number of nodes in order to evaluate the scalability of the system. As other simulation parameters, we define O and T, which are the number of operation executions and the number of topologies, respectively. In order to measure the traffic overhead of an operation for a specific N, T random topologies are generated. On each of these T random topologies, the operation of which the traffic overhead is being measured, is executed O times. In each of these O executions, nodes that are involved in the operation are randomly selected. At the end, all $T \times O$ experiments are summed up and they are divided by $T \times O$. In result, arithmetic mean is obtained, which is named as the "average traffic overhead". One last simulation parameter we define is S, which is the type of the operation (e.g. *Insert*) of which the traffic overhead is being measured. All the simulation parameters are listed in Table 6.1.

Table 6.1: The simulation parameters

N	: the number of nodes
O	: the number of operation executions
T	: the number of topologies
S	: the operation type

We designed the simulation environment to be composed of three fundamental components (See Figure 6.1). The *Simulation* component provides interface to the user and it is used for controlling the simulation parameters and obtaining the results. The *Engine* component supports the functionality of underlying layers on which our system runs. It handles intercommunication of nodes, informs about node discovery and notifies link failures. On top of this component, multiple copies of the $node_x$ $(0 \leq x < N)$ component runs, each of which operates according to protocols defined in the previous chapter.

At the very beginning of the simulation, N number of nodes are created and their locations on a plane are randomly determined. The *Engine* component, which is shared by all nodes, constructs a *range matrix*, based on the locations of nodes. By means of this binary matrix, the *Engine* knows which nodes are in the communication range of each other. Whenever a node wants to send a message to another node, it assembles a message and enqueues it to the *message queue* of the *Engine*. The *Engine* continuously dequeues messages and sends them to their destinations after checking the feasibility from the *range matrix* and incrementing the *message counter*, which is outputted at the end.

Each node runs as an independent thread. It parses messages dequeued from the *incoming message queue*, operates in accordance with protocols of our system, assembles and sends messages through the *outgoing message queue*, if necessary. Meanwhile, it obtains nodes that are in the communication range of itself from the *Engine* and tries to connect them, if they have not the same network ID. It may also receive interrupts from the *Engine* about link failures (i.e. updates in the node-related entries of the *range matrix*), after which the *Recover* operation is initiated.

For the construction of each topology, nodes are replaced randomly on a plane. Since nodes tend to connect other nodes within their communication range, the network is formed in a self-organizing manner. When all nodes have the same network ID, the *Engine* realizes that the network formation is completed. It is also possible that two or more disjoint networks are formed, in which any two members of different networks are not within the communication range of each other. However, all node pairs that are within their communication range must have the same network ID, before we can affirm that the network formation is completed. In our simulation, we make dense deployments in order to prevent formation of disjoint networks. After network formation is completed, a specific operation is executed O times and the messages exchanged during each execution of the operation are counted. Issues related to how and which operations are tested, which values are set as simulation parameters, how nodes that involve in an operation is chosen and results are explained in the following section.

The simulation environment is implemented with Java^{TM} language. Java 2 SDK, Standard Edition is used as the development environment. Only standard libraries provided by Java 2 SDK, Standard Edition (version 1.4.1) is used.

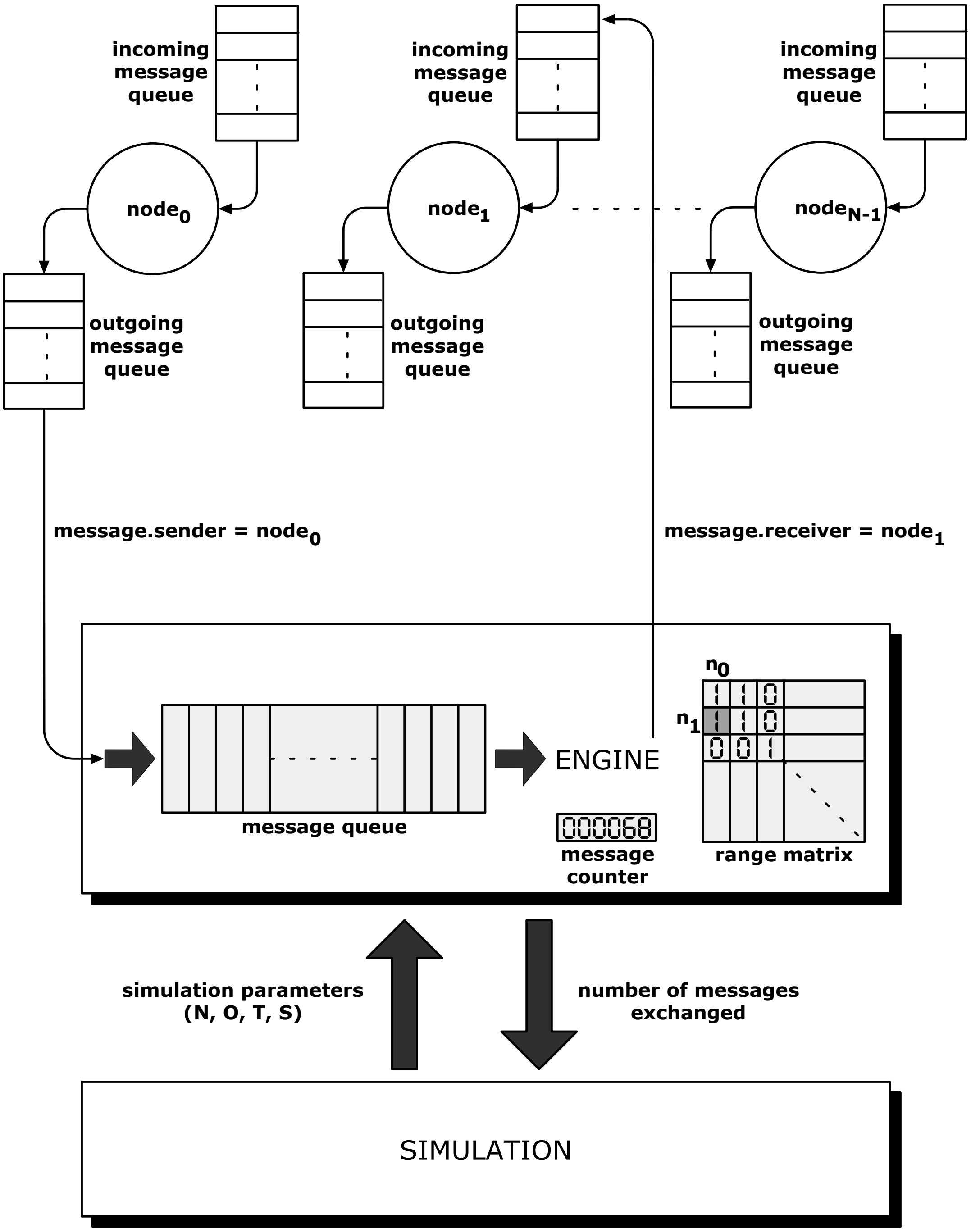


Figure 6.1: The simulation environment.

6.2 Results and Evaluation

In this section, we present and comment on results of experiments that we performed on the simulation environment that is explained in the previous section. We have evaluated the traffic overhead of the *Insert*, *Access2F-Node*, *Recover* and *Network-Join* operations in varying number of nodes. We did not evaluate all operations since, traffic overhead of some operations are very related to each other, if they are not the same. The *Insert*, *Delete* and *Access2P-Node* operations, for instance, have the same traffic overhead. This is because, other than the actions of the *Acce2P-Node* operation that is being used of, actions of the *Insert* and *Delete* operations, do not involve any messaging.

During all measurements, both T and O are set as 100. In other words, for each operation and for each specific N, 10000 experiments are performed and their arithmetic mean is computed as the average traffic overhead. For each operation, average traffic overhead measurement is performed for values of N starting from 10 and increasing up to 100 with increment amount being 10.

In order to measure the traffic overhead of an execution of the *Insert* operation, a randomly selected node is made to initiate the *Insert* operation regarding a randomly selected P value, which is in reality would be the hash value of a file name. Messages exchanged until the completion of the operation is counted, thereafter. In Figure 6.2, results we obtained are presented, in which the average number of messages exchanged is shown with varying number of nodes starting from 10, increasing up to 100. We can observe a linear increase in the average traffic overhead with increasing number of nodes. As briefly discussed in the previous section, the number of messages exchanged during the execution of the *Access2P-Node* operation is $N-1$, in the worst case and the *Insert* operation has the same traffic overhead that of the *Access2P-Node* operation. So, even in the worst case, traffic overhead of the *Insert* operation is subject to a linear increase with increasing number of nodes. Actually, since our system makes use of a tree shaped overlay network, the average traffic overhead of the *Insert* operation is related with $log_m n$, where m is the node degree. However, m differs for each generated topology and even for each node of a topology. That is why, results leaded to a linear increase but a less steeper curve compared to the worst case.

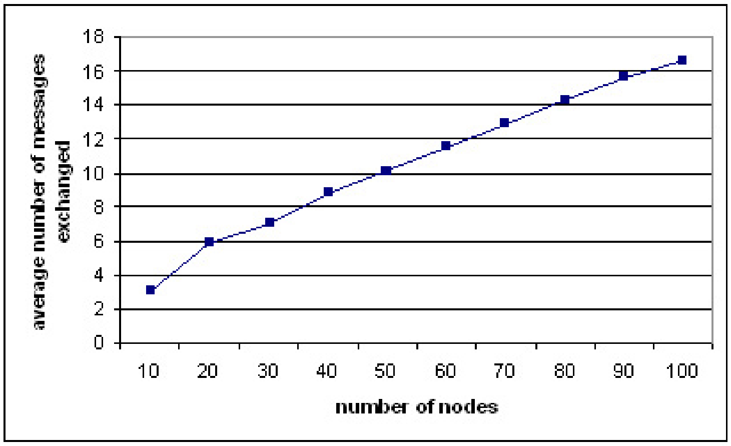


Figure 6.2: The average traffic overhead of the *Insert* operation.

Analogous to the traffic overhead measurement of the *Insert* operation, for performing an experiment concerning the *Access2F-Node* operation, a randomly selected node is made to access a file regarding a randomly selected P value. Beforehand, a randomly selected node is made to complete an *Insert* operation regarding the selected P value, so that it is ensured that such a file, thus a corresponding *F-Nod*e exists to be accessed. Messages exchanged starting from the initiation of the *Access2F-Node* operation until the completion of this operation is counted. Figure 6.3 illustrates experiment results. As was the case with the results of the *Insert* operation, a linear increase in the average traffic overhead with increasing number of nodes can be observed. This is because; also the *Access2F-Node* operation basically relies on the *Access2P-Node* operation. When the *Access2P-Node* request is initiated with *Access2F-Node* as the request type, *P-Node* is reached at first. From *P-Node*, location information is returned back to the source of the query. Having location information, source of the query accesses *F-Node* (Recall Figure 5.2). When the *Access2P-Node* request is initiated with *Insert* as the request type on the other hand, *P-Node* is reached and location information recorded in the request is stored at the destination. No extra messages are exchanged. This can be verified in Figure 6.4, in which the number of messages exchanged during the execution of the *Insert* and *Access2F-Node* operations

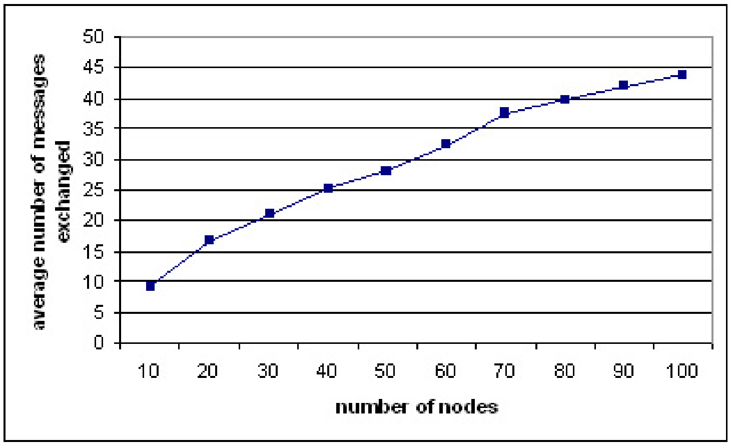


Figure 6.3: The average traffic overhead of the *Access2F-Node* operation.

are compared. As it can be observed from the figure, the number of messages exchanged for the completion of the *Access2F-Node* operation is approximately 2 times more than the number of messages exchanged for the completion of the *Insert* operation.

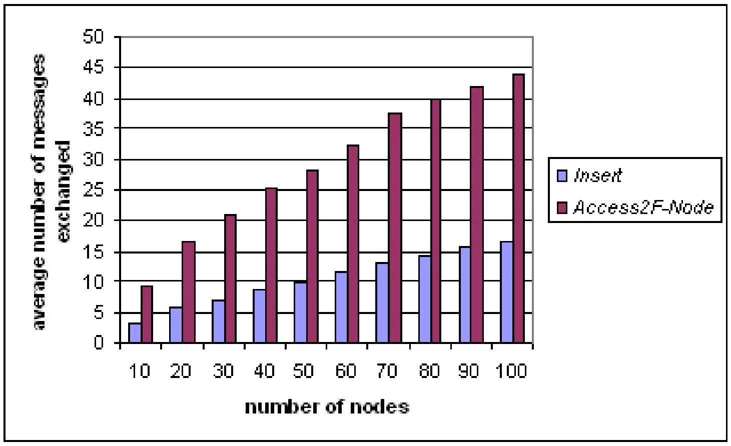


Figure 6.4: The average traffic overhead comparison of the *Insert* and *Access2F-Node* operations.

In order to measure the traffic overhead of the *Recover* operation for a specific

N, a network is formed randomly with N number of nodes. After network formation is completed, a connection between a pair of nodes, which are randomly selected, is disabled, on purpose. Simply setting the corresponding entry in the *range matrix* to 0 does this. Eventually, the *Engine* notifies corresponding nodes about the link failure, after which both nodes automatically initiate the *Recover* operation. Messages are counted until the reconfiguration is completed and two separated networks are stabilized. Consequently, the overhead counted is not just for the recovery of a disconnected sub network but the whole network. Before next experiment on the same topology is performed, the modified entry in the *range matrix* is switched back to 1 and completion of the proceeding *Network-Join* operation is waited to obtain the original topology back. In Figure 6.5, average traffic overhead of the *Recover* operation is presented with varying number of nodes and again, a linear increase in the overhead is observed. Nevertheless, this

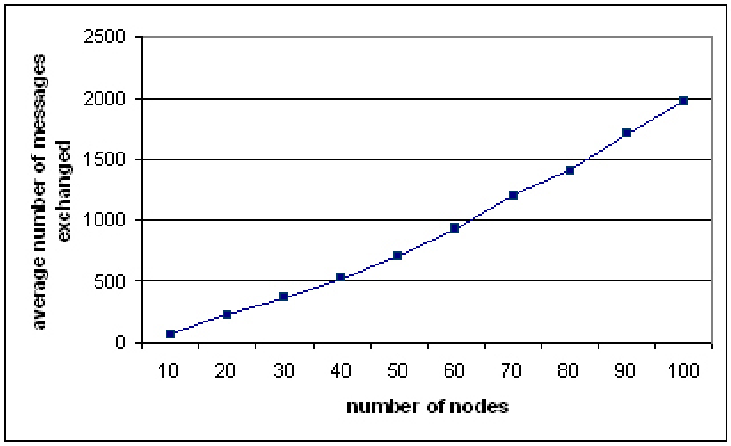


Figure 6.5: The average traffic overhead of the *Recover* operation.

time, the overhead is much higher compared to the results regarding the *Insert* and *Access2F-Node* operations. This is not surprising however since, during the execution of the *Recover* operation, *hashline* is redistributed in the whole sub network, where disconnected child node becomes the root. Meanwhile, many *Insert* operations are initiated in order to regain the lost information in the base network. The number of *Insert* operations called, depends on the length of the *hashline* segment that is lost and the number of files shared in the system. The first parameter is randomly determined since the disconnection point in the network is randomly selected. When it comes to the number of shared files, in our simulation, we make each node to share x number of files, where x is a random integer between 1 and 10. Since, files are mapped on the *hashline* through a uniform hash function, they are supposed to be evenly distributed on the *hashline*.

Traffic overhead of the *Network-Join* operation is measured in the same way that of the measurement of the *Recover* operation. This time, however, messages exchanged during the completion of the *Network-Join* operation are counted after recovery of two disjoint networks are accomplished. The results we obtained are presented in Figure 6.6. Apparently, the *Network-Join* operation has the most traffic overhead among all operations, which is even much higher than the overhead of the *Recover* operation. Moreover, in Figure 6.6, the average traffic overhead appear to increase exponentially with increasing number of nodes. The excessive number of *Insert* operations executed would cause this high cost. Recall that each node in the joining network must send *Insert* requests concerning the whole network for each file it shares (Each node shares x number of files, where x is a random integer between 1 and 10). Other than this, there is an overhead of the redistribution of the *hashline* as well but, that was also an issue for the *Recover* operation and it did not lead to an exponential increase in the average traffic overhead.

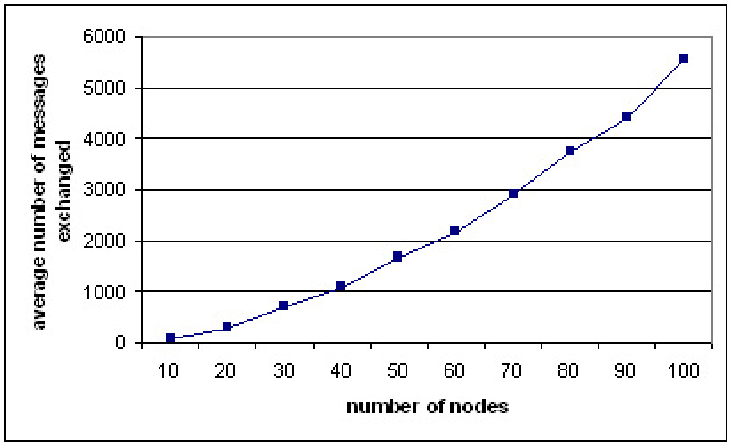


Figure 6.6: The average traffic overhead of the *Network-Join* operation.

In Figure 6.7, results regarding all of the four operations are compared. Traffic overheads of the *Insert* and *Access2F-Node* operations are hardly recognized along with the overhead of other two operations and especially the high traffic overhead of the *Network-Join* operation is noticeably observed.

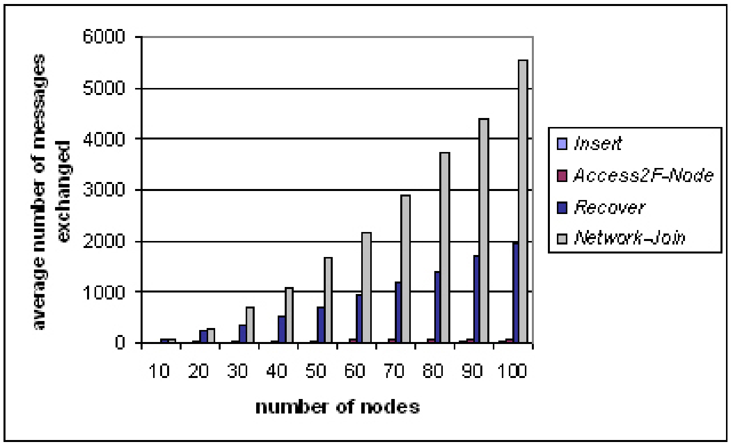


Figure 6.7: The average traffic overhead comparison of the four operations.

As the results indicate, our system enables efficient access to shared files. However, it may not work efficiently when frequent disconnections occur, which would lead to a high traffic overhead. Nonetheless, we believe that the environment in which a file sharing system would be used is a WANET where mobility should be supported, but the rate of mobility is not high. A conference room may be given as an example to such an environment, in which attendees need to share files. Although the mobility rate is not high, WANET should support the mobility and ad-hoc features, where there is no infrastructure support and the network is build up upon demand for a relatively short duration of time.

If members of the file sharing enabled WANET continuously move around but they very infrequently search for files, then flooding would be a better approach to apply. In our system, we try to preserve consistent and distributed location information, in order to prevent flooding and access files through unicast queries. The cost we paid for keeping the location information consistent would be amortized by the exceedingly reduced cost of subsequent file searches. When such information does not exist, each file search performed by each node of the network would flood the whole network and lead to a congestion. Members of the file sharing enabled WANET can be mobile, but we presume that more than they move around, they perform file searches.

The entire set of experiment results on which charts presented up to now are based on, are listed with exact values in Table 6.2.

Table 6.2: The average number of messages exchanged until completion of operations.

the number of nodes	*Insert*	*Access2F-Node*	*Recover*	*Network-Join*
10	3.0804	9.2076	68.4657	94.5506
20	5.8848	16.6844	236.4135	299.8259
30	7.0843	21.1111	362.8687	693.0449
40	8.8213	25.27	529.8643	1095.358
50	10.1681	28.233	707.3418	1669.999
60	11.5373	32.4310	927.4455	2189.172
70	12.8763	37.5278	1194.971	2908.522
80	14.3166	39.6046	1407.703	3744.495
90	15.6217	41.8603	1708.507	4403.113
100	16.5613	43.9152	1967.679	5571.575

Chapter 7

Conclusion and Future Work

In this thesis, we proposed a peer-to-peer system that enables file sharing in wireless ad-hoc networks. File sharing systems that are used today are specialized for wire-line networks, namely the Internet. Although they introduce neat solutions for the file sharing problem in general, they make use of routing functionalities that are handled by the underlying networking layers. This is not possible in wireless networks, where each node is only aware of its surrounding, i.e. the nodes in its range of communication. Although there have been solution proposals for the routing problem in WANETs, there is no widely accepted, standard routing protocols for this type of networks. File sharing systems proposed specifically for WANETs up to now, on the other hand, are based on flooding, which leads to a substantial traffic overhead.

The novel approach introduced in this work is the unification of lookup functionality and routing functionality, which results in a cross-layer scheme. The system keeps track of the routing information together with the location information, which is fully distributed. In achieving this, we adapt techniques from peer-to-peer systems developed for wire-line networks as well as source routing techniques. By means of constructing a distributed hash table (DHT) and forming a tree shaped overlay network based on the topology of the network, the system is able to find the location of a file in a WANET, if such a file exists in any node of the network, and it finds a way to bring the file from where it is stored to where it is needed. Only three functionalities should be supported by the underlying WANET protocols, which are about handling communication between any two nodes that are in the range of each other. Multi-hop routing of information is handled by the peer-to-peer system itself. Our system also handles disconnections and reconnections that may happen as a result of mobility or due to problems in the wireless channel.

Simulation results showed that our system efficiently carries out file searches and it is also scalable. However, in the case of frequent disconnections, the system would introduce too much traffic overhead on the network. Consequently, we can state that our system works better on a WANET with low rate of mobility and frequent file searches. In the opposite case, flooding may lead to better results, in which no information is maintained and network is flooded when necessary (i.e. a file search is performed).

As a future work, another simulation environment can be developed aiming to compare overhead of flooding and our system in varying rate of mobility and frequency of file searches. By this way, we can be able to observe above which *file search frequency/rate of mobility* ratio as a threshold, our system performs better and vice versa.

Also, the *Recover* and *Network-Join* operations can be modified, so that they have less traffic overhead. At least, nodes can wait for some time when disconnections occur, hoping that the connection will be retained before a timeout is exceeded. By this way, intermittent disconnections for short time periods would not lead to frequent execution of *Recover* and *Network-Join* operations one after the other.

A supplementary modification is possible in the hash function that is used to map file names to certain keys (i.e. points on the *hashline*). In our system, we propose the usage of any uniform hash function for that purpose. This approach has a disadvantage that only search of exact file names are possible. Instead of a uniform hash function, other mapping techniques can be used like Soundex [7], which maps similar names to same keys. In that case, a set of results will be returned as an answer to queries.

Responsibility sharing policy can also be reconsidered. Currently, in our specifications, the *hashline* segment is divided into two halves, when it is going to be shared. As an alternative, the dissection point can be determined according to the distribution of files, although this may change in time. Such an approach can especially be useful when a uniform hash function is not used as discussed in the previous paragraph.

As a final point, we would like to implement the system and make it practically work in a real environment, a Bluetooth scatternet of a set of pocket PCs, for instance.

Bibliography

[1] S. Androutsellis-Theotokis. A survey of peer-to-peer file sharing technologies. Technical Report WHP-2002-003, ELTRUN, Athens University of Economics and Business, 2002.

[2] Bluetooth Special Interest Group. *http://www.bluetooth.com*.

[3] Fasttrack. *http://www.fasttrack.nu*.

[4] Gnutella. *http://gnutella.wega.com*.

[5] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad-hoc wireless networks. In T. Imielinske and H. Korth, editors, *Mobile Computing*, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[6] A. Klemm, C. Lindemann, and O. Waldhorst. A special-purpose peer-to-peer file sharing system for mobile ad hoc networks. In *Proceedings of IEEE Semiannual Vehicular Technology Conference (VTC2003-Fall)*, 2003.

[7] D. Knuth. *The Art of Computer Programming*, volume 3 (Sorting and Searching). Addison-Wesley, second edition, 1998.

[8] C. Lindemann and O. Waldhorst. A distributed search service for peer-to-peer file sharing in mobile applications. In *Proceedings of 2nd IEEE Conference on Peer-to-Peer Computing (P2P 2002)*, pages 73–81, 2002.

[9] D. A. Menasce. Scalable P2P search. *IEEE Internet Computing*, 7(2):83–87, 2003.

[10] Napster. *http://www.napster.com*.

[11] M. Papadopouli and H. Schulzrinne. Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices. In *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 117–127, 2001.

[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM Conference*, pages 161–172, 2001.

[13] E. Royer and C. K. Toh. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Communications*, 6(2):46–55, 1999.

[14] 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 the ACM SIGCOMM Conference*, pages 160–177, 2001.

[15] B. Y. Zhao, J. D. 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, U. C. Berkeley, 2001.

Appendix A

Table of Acronyms

CAN	Content-Addressable Network
DHT	Distributed Hash Table
DSR	Dynamic Source Routing
IP	Internet Protocol
MAC	Medium Access Control
PDA	Personal Digital Assistant
P2P	Peer-to-Peer
SDK	Software Development Kit
WANET	Wireless Ad-Hoc Network