A Framework for Peer-to-Peer Computing in Mobile Ad Hoc Networks

by

Afzal Mawji

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Doctor of Philosophy

Queen's University
Kingston, Ontario, Canada
January 2010

Abstract

Peer-to-peer (P2P) applications are enormously popular on the Internet. Their uses vary from file sharing to Voice–over–IP to gaming and more. Increasingly, users are moving toward wireless networked devices and wish to continue using P2P applications in these new environments. A mobile ad hoc network (MANET) is an infrastructureless network which allows users to dynamically form a mobile, wireless network. Though P2P and MANETs share some similarities, such as self-organization, dynamism, and resilience to failure, it is necessary to create new P2P algorithms that take advantage of the realities of MANETs. These algorithms must account for the numerous challenges found in these networks, including node mobility, resource constrained nodes, and the necessity of fully distributed algorithms.

In this thesis, we propose a framework for mobile P2P computing in MANETs (P2P–MANETs). Our proposal includes the following components. First, nodes must be able to locate and join the P2P overlay. We therefore propose a fully distributed bootstrapping algorithm in which nodes multicast join requests and cache responses. Next, the overlay peers must form a topology of connections between themselves. We propose a fully distributed topology control heuristic which supports the dynamic nature of the P2P–MANET. It is important that peers contribute to the network by sharing their resources and forwarding traffic for others. We therefore propose

a dynamically priced incentive scheme which rewards users for contributing to the network. We also propose a path selection algorithm to allow peers to select how many parts of a file to download from which servers and which paths to satisfy the user's preference for download time and cost. Finally, we propose a content distribution system that allows users to download large files through the use of network coding and multicasting. Each of these components is the first proposed for its respective place in a P2P–MANET architecture. Simulation results show that each of the proposed components achieves the goals set out for it and outperforms the comparison schemes. The results also show that the overlay topology and path selection heuristics provide good approximations compared to the optimal solutions.

Co–Authors

- A. Mawji, H. Hassanein, "Efficient Content Distribution for Peer–to–Peer Overlays in Mobile Ad Hoc Networks", *journal paper in preparation.*
- A. Mawji, H. Hassanein, "Bootstrapping Peer–to–Peer Overlays in Mobile Ad Hoc Networks", *journal paper in preparation.*
- A. Mawji, H. Hassanein, "Incentives to Contribute to Peer–to–Peer Overlays in Mobile Ad Hoc Networks", *journal paper submitted for publication.*
- A. Mawji, H. Hassanein, "Peer–to–Peer Overlay Topology Control for Mobile Ad Hoc Networks", *journal paper submitted for publication.*
- A. Mawji, H. Hassanein, "Optimal Path Selection for File Downloading in P2P Networks on MANETs", *conference paper submitted for publication.*
- A. Mawji, H. Hassanein, "P2P Overlay Topology Control in MANETs", *conference paper submitted for publication.*
- A. Mawji, H. Hassanein, "Implementation of Bootstrapping for P2P Overlays in MANETs", *conference paper submitted for publication.*
- A. Mawji, H. Hassanein, "Efficient Multipoint P2P File Sharing in MANETs", *IEEE Global Telecommunications Conference (GLOBECOM)*, 2009.
- A. Mawji, H. Hassanein, "Incentives for P2P File Sharing in Mobile Ad Hoc Networks", *IEEE Consumer Communications and Networking Conference (CCNC)*, 2009.
- A. Mawji, H. Hassanein, "Bootstrapping P2P Overlays in MANETs", *IEEE Global Telecommunications Conference (GLOBECOM)*, 2008.
- A. Mawji, H. Hassanein, "A Utility–Based Incentive Scheme for P2P File Sharing in Mobile Ad Hoc Networks", *IEEE International Conference on Communications (ICC)*, pp. 2248–2252, 2008.

Acknowledgments

I am grateful to my supervisor, Dr. Hossam Hassanein, for giving me the opportunity to pursue my Ph.D. and for providing a supportive environment in which to do so. His knowledge, guidance, support, motivation, and faith in my abilities were instrumental throughout the course of my research.

I would also like to thank my wife, Manisha, and rest of my family and friends for their support and good wishes. Their encouragement helped to get me through the more trying times and I enjoyed celebrating even the smallest milestones with them.

Thanks to Xiangyang Zhang for his comments on this thesis.

My friends and colleagues in the Telecommunications Research Laboratory, especially Ahmed Hasswa and Kashif Ali, gave me great support and fruitful discussion, as well as some sporting enjoyment. I will miss the convivial environment of the lab.

I would also like to thank my Ph.D. committee members, Dr. T.C. Nicholas Graham and Dr. Mohammad Zulkernine, for their valuable feedback throughout all stages of my Ph.D. candidacy. Thanks also to my Ph.D. defence committee members, Dr. Azzedine Boukerche, Dr. Jim Cordy, and Dr. Chi–Hsiang Yeh.

Finally, I would like to acknowledge the financial support provided by Queen's University and the Natural Sciences and Engineering Research Council of Canada (NSERC).

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Wireless mobile services provide convenient access to information and services, including mobile commerce, geographical and location information, Web services, cooperative group work, streaming media, voice, content sharing, instant messaging, and gaming. The services should be available to users wherever they are, and whenever they are needed.

A traditional mobile computing environment consists of wall–powered, resource–rich, stationary servers communicating with a number of clients over a fairly predictable wireless link via a network access point such as a base station. The client/server paradigm is predominant and mobile clients do not communicate directly with one another.

In a multi–hop, variable topology network, such as a mobile ad hoc network (MANET), a pre–existing infrastructure is not needed. Instead, devices form a self–organizing network with their peers and communicate with each other in multi–hop "peer–to–peer" mode. Any node may act as a client and/or server at any time, so

both clients and servers are mobile and resource–constrained. Devices are heterogeneous, resulting in a potentially large variation in link and node capabilities, and due to mobility, the network topology is constantly changing. These autonomous and infrastructureless networks pose particularly challenging design problems, especially with regard to lifetime and scalability.

Infrastructureless networks such as MANETs are essential in some environments, such as in disaster recovery, or military battlefields, where the communications infrastructure has been damaged or is non–existent. In other environments, such as at a company meeting where the participants would like to share documents, such networks may prove more convenient than using a wireless LAN. MANETs are used in combined infrastructure–infrastructureless networks such as wireless mesh networks and they are also expected to become an important part of the 4G architecture [10]. Ad hoc nodes will use multi–hop communication to provide wider networking coverage and to connect to the fixed–backbone network, thus providing Internet services to areas without pre–existing infrastructure. Recently, the Wi–Fi Alliance has announced a new specification called "Wi–Fi Direct" that allows devices to connect to one another directly, without the use of a base station [84]. The code–name for the specification was "Wi–Fi peer–to–peer".

The client/server model allows the powerful server to perform processing and to provide services to a weaker client device. As more clients require service, the additional load may make the server a bottleneck and prevent it from performing its duties successfully. An alternative, to the client/server model is the peer–to–peer (P2P) model. The term "peer–to–peer" was first used to describe point–to–point communication between two nodes of equal status, similar to a telephone conversation

[81]. It is now considered a communication paradigm for large numbers of nodes, with some nodes possibly more equal than others. P2P networks focus on using the resources of the peers, such as storage, CPU cycles, content, and bandwidth. These resources are shared with other peers so that idle resources may be used for a greater benefit. Peers still use the client/server paradigm in the sense that any time one peer contacts another it is acting as the client, while the other acts as a server. In fact, as a peer, a node acts as *both* a client and a server. The major benefits of P2P are its leverage of previously unused nodal resources, potentially faster information delivery due to removal of the server bottleneck, cost savings due to a reduced need for centralized management, storage, and other related resources, easier scalability, and increased network fault tolerance. Drawbacks of P2P include lack of guarantees on availability and performance, lack of control of the network, possible loss of privacy, and security issues.

P2P networks are usually implemented in the form of overlay networks, which consist of higher–layer connections between peers that are independent of the underlying or substrate network. This allows overlay networks to abstract their connectivity to a higher–level view of the peers that make up the network. The overlay is formed by the amalgamation of the logical connections that peers form with one another. Peers communicate with one another via the overlay, with queries and responses possibly being routed over multiple hops. The peers self–organize and must be able to handle "churn", the term used for the constant connections and disconnections of peers as they join and leave the network, meaning P2P networks must be fault tolerant. On the Internet, peers exist on the access links, or "edge" of the network, and many P2P networks in use today scale to millions of simultaneous users.

P2P systems may be categorized into centralized, brokered, and decentralized systems [81]. Centralized systems require peers to connect to a server, which coordinates and manages all communication. Brokered systems require peers to connect to a server to discover other peers, but peers manage communication between themselves. In decentralized systems, there are no servers; peers operate independently and perform both discovery and communication in a decentralized manner.

Peer–to–peer (P2P) networks are increasingly popular and their uses vary over many different application types, such as file sharing, Voice–over–IP, video streaming, gaming and instant messaging. Boukerche *et al.* [12] recently proposed the use of a P2P overlay on a MANET using Gnutella [37] to support Mobile Collaborative Virtual Environments, 3D graphic applications in which mobile users interact and collaborate in a virtual environment.

It is clear that users want to run P2P applications, regardless of what type of network they are using, be it a wired network, a MANET, or a hybrid infrastructure–infrastructureless network such as a wireless mesh or 4G network. Therefore, it is important to consider how to effectively run a P2P overlay in an infrastructureless network such as a cooperative MANET. If the user is connected to a partly–infrastructured network, avoiding the base station and instead communicating with nearby nodes may reduce delay and energy use since transmissions travel a shorter distance and require less power. Furthermore, avoiding communication with the base station may avoid charges from the network operator. Finally, if there are multiple infrastructured networks in an area, such as a university campus, it may be possible for users of these different networks to join together to form large, cooperative MANET.

It is a natural evolution for MANETs and P2P networks to be combined together so that a P2P overlay runs on a cooperative MANET (we use the term P2P–MANET to refer to such networks), since both types of networks have many similarities. Both are decentralized networks, both must dynamically organize themselves, both must deal with frequent topology changes, both attempt to be resilient to failure, and both perform the routing function.

Despite the similarities between P2P networks and MANETs, it is unclear that simply adopting existing P2P overlay techniques and using them in MANETs is desirable, since there are also differences. P2P networks tend to be very large–scale with millions of simultaneous users, and are designed as overlays for deployment on the "edge" of the Internet, where the nodes generally do not move about. On the other hand, MANETs tend to have far fewer nodes, the devices are severely resource–constrained in comparison, and the links between nodes usually have higher delay. Energy consumption is of great concern and users are also geographically nearby one another.

Figure 1.1 illustrates an example of a peer–to–peer overlay on a MANET. The light circles represent nodes participating in the overlay network, while the dark circles are not part of the overlay. The solid lines show how the overlay network is connected, with potentially multiple hops of the underlying MANET providing the direct overlay connections. The dashed lines represent the MANET links.

1.1 Motivation and Objectives

We expect that in the future, as wireless networks with mobile ad hoc components such as mesh and 4G networks begin to see large increases in the number of users,

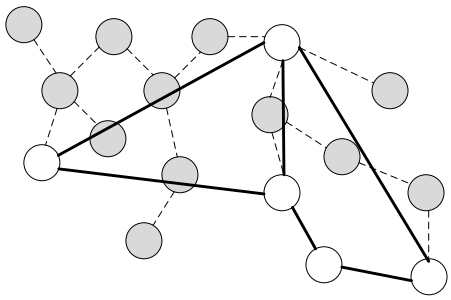


Figure 1.1: An example of a peer–to–peer overlay running on a MANET

those users will continue to show a preference for sharing information in the form of P2P networks. It is therefore necessary that a framework for P2P–MANETs exist, to enable these networks to function.

The majority of peer–to-peer computing research has focused on issues related to efficient lookup and routing. This has led to the creation of distributed hash tables (DHTs), in which data is stored on particular nodes of the overlay, according to the results of a hash function. When a search must be done, the hash function quickly reveals the location of the data. The most efficient algorithms have lookup times of approximately $O(\log N)$ for N peers, but require a specific topology to be enforced. In both wired and wireless networks, it is better if the overlay topology reflects the physical layout of the underlying nodes, otherwise a single overlay hop may equate to the diameter of the network. This would increase delay and also, in MANETs, energy consumption.

Other important issues in P2P computing include bootstrapping, the process of locating and joining a P2P overlay; incentive mechanisms, which encourage users to participate in the P2P network; server and path selection, the process of selecting which of several potential servers and paths to choose to obtain services from; and efficient content distribution, in which large content such as a file is obtained by

interested peers. All of these issues require separate consideration when the P2P network is implemented over a MANET. Since there is no centralized node, existing boostrapping solutions, which require contacting a list of "well–known" nodes, will not work. Existing topology control schemes ignore the substrate, or underlay network, which may lead to excessive delay due to a large underlay hop distance while also increasing energy use. Incentive mechanisms in P2P–MANETs must persuade users to not only share content and services, but also to forward data along the multi–hop path to the final destination. Selecting a set of servers and the paths to them should account for important factors such as delay and energy consumption. Large files must be downloaded efficiently in P2P–MANETs to reduce energy consumption and download times for greater user satisfaction. Existing research on P2P–MANETs focuses mainly on routing issues, typically proposing cross-layer protocols to merge P2P lookup with MANET routing.

For the most part, existing P2P algorithms cannot be directly applied to MANETs due to the lack of infrastructure, node mobility and energy issues. For example, Cramer and Fuhrman [24] examine how to bootstrap the well–known P2P DHT Chord [78] in an ad hoc network. They show that the process has a time complexity that grows linearly with network size and creates a significant traffic load imbalance at the join point. Therefore, it is important that existing P2P algorithms be redesigned to take into account the substantially different environment of MANETs.

Energy consideration is always an important goal when designing protocols for MANETs. Each packet that is transmitted consumes energy on every node that the packet passes through, so reducing the number of messages has a direct effect on energy consumption. Furthermore, reducing the number of packets also reduces

contention for the shared wireless medium, which has positive effects on the delay. Therefore, an effective P2P–MANET system will consider energy use throughout all its components.

The infrastructureless nature of MANETs is a crucial consideration in the design of effective P2P–MANET protocols. Although a pure P2P network does not require any infrastructure, few such networks exist in practice. P2P networks typically require infrastructure at several points during their execution, particularly during bootstrapping. Bootstrapping typically requires some form of centralization or infrastructure, such as a list of well–known nodes that are usually online, or a "recently–online" node list downloaded from a Web server. In a completely infrastructureless network, such bootstrapping methods are impossible to use.

Nodes in MANETs are mobile, meaning that the underlying topology is constantly changing as users move about. Therefore, it is important that the overlay topology reflect the underlying network, both to reduce energy consumption and to improve response times. Existing P2P topology control schemes, designed for non–mobile wired networks, are able to accomodate a changing topology due to the expectation that nodes will constantly be joining and quitting. However, nodes are assumed to be disconnected once they are unreachable. In MANETs, when a node is no longer reachable, it is incorrect to assume that the node has left the network, since it may in fact have simply moved to a different location. A topology control mechanism for P2P–MANETs must efficiently reconfigure the overlay topology to account for the change in the underlay network.

Both MANETs and P2P networks require nodes to help one another in order to make the network useful. P2P file sharing networks require nodes to share files,

and MANETs require nodes to forward data. Unfortunately, helping others comes at a cost. By sharing files and forwarding data, mobile nodes are using their energy, bandwidth, CPU, and other resources. When resources are used by others, there is less available for the node to use for itself. This is especially important for resources that are not easily renewed such as energy.

MANET nodes that do not forward are referred to as *selfish nodes* and P2P nodes that do not share are called *freeloaders* or *free riders*. In a typical P2P network the majority of nodes, generally between 66% and 90%, are freeloaders, and only a small percentage are altruistic, meaning that they share data unconditionally [1]. To entice users to share files and forward data, many incentive schemes have been proposed. The basic idea behind such schemes is to encourage users to contribute files and forward data for others by rewarding users who do so and punishing those who do not. For example, if a selfish node chooses not forward data, this information can be propagated to other nodes in the MANET, who will then actively ignore requests to forward on behalf of the selfish node. A similar idea can be used in P2P networks, though it may be more difficult to ascertain whether a node is freeloading or not. An effective incentive scheme for P2P–MANETs would produce both the effects of encouraging users to forward traffic and also to share files. One way to accomplish this is to introduce the concept of virtual credits. Nodes would be rewarded with credits for sharing files and forwarding data, and when they opt to download files, they would be required to spend the credits they have earned.

When a user wishes to obtain a service from the P2P network, such as downloading a file, it must execute a query. Once the results are back, there may be several possible servers willing to offer the file. The P2P–MANET system must be able

to select the best servers and the best paths to those servers, taking into account the cost of downloading the file, which may include factors such as delay, energy consumption, and popularity of the file. If the user is able to download parts of the file from multiple servers simultaneously, this would reduce the download time. It is important therefore, for the client node to determine how much to download from valid paths such that the resulting selections meet the objectives of a low cost and download time.

The most popular types of P2P networks distribute content. In a P2P–MANET, if users are attempting to download large files it will create very large amounts of network traffic, resulting in congestion and high energy consumption. It is desirable for peers to cooperate with one another in such a manner so as to reduce the amount of network traffic required. This can be accomplished by reducing the amount of coordination needed to locate content, such as is the case with network coding, and also by employing a technique, such as multicasting, which reduces the number of transmissions required in the first place.

To date, there is no existing P2P–MANET system that addresses all of the above–mentioned issues and some have not been addressed at all. In this thesis we address these shortcomings in existing P2P–MANET research. Our research is intended to further the goal of a complete P2P–MANET system. Specifically, we propose a P2P–MANET framework, consisting of a distributed, energy efficient bootstrapping technique for P2P–MANETs, a distributed, computationally efficient overlay topology control scheme that reflects the underlay network, an incentive mechanism that considers energy consumption and delay, a server path selection algorithm, and an efficient content distribution system which permits faster downloads while also using

less energy. We provide the framework for these components, in order to allow a complete and practical P2P–MANET system to be implemented. These components are developed independently of one another and may be used as such, or they may be combined together.

Throughout this thesis, we use the term MANET node to refer to a device in the MANET that is not a part of any P2P overlay. This includes devices that are in the process of joining an overlay. The terms node, peer, and overlay member are used interchangeably and refer to a device in the MANET that is part of at least one P2P overlay. Peers that form overlay connections to one another are called neighbours. The collection of neighbourhood relationships forms the overlay graph. Clients are nodes that are making a request, and servers are peers that respond to requests and perform a service. Clients may simultaneously be acting as servers for other peers, and servers may simultaneously be clients as concerns other overlay members.

1.2 Thesis Contributions

P2P computing and mobile computing are immensely popular today, but the two have generally not been used together. We envision general–purpose MANETs with many applications and services atop them becoming more common, particularly as they are joined to mesh and 4G networks. In that event, we believe that one application that will be enormously popular will be P2P–MANETs. Therefore, we propose to provide a framework for P2P–MANETs consisting of the necessary components to allow practical P2P–MANETs to be implemented. The framework consists of the following components:

1. A bootstrapping algorithm that does not require the existence of an infrastructure. Current bootstrapping techniques require either infrastructure or the existence of "well–known" nodes that are usually online. In a P2P–MANET, neither infrastructure nor long–lived nodes may be available, and it is therefore important that nodes are able to identify and join the P2P overlay without them.

2. A fully decentralized, computationally feasible overlay topology control algorithm that is aware of the underlying network. The nodes of the overlay are connected together using logical links and the topology of the overlay affects the performance and energy use of the overlay. Therefore, a topology that considers the underlying physical network will tend to be most efficient. Furthermore, since nodes will be joining and leaving the overlay, and also moving around within the network, it is important that the algorithm be able to quickly determine an efficient topology.

3. An incentive mechanism that provides pricing for sharing files and forwarding data. An incentive scheme rewards users for sharing their resources and promotes cooperation within the network. Peers use credits when they consume services within the network, and earn credits for contributing to others within the network. These contributions can take the form of providing services, such as sharing files, or acting as an intermediate node that forwards data on behalf of others.

4. A path selection algorithm that chooses how much to download from several paths in order to obtain the lowest download time subject to a budget. By

choosing the lowest download time paths and servers, a peer can obtain services with the lowest possible delay. This algorithm will allow users to download from multiple servers simultaneously, if possible. For example, it will allow a node to download parts of a file from different paths and servers at the same time, reducing the download time and cost subject to a user preference parameter. The algorithm indicates how much to download and from whom, but does not include the actual task of downloading.

5. A content distribution system that allows users to efficiently download large files. A large proportion of P2P traffic is the sharing of files, so a means by which the traffic can be reduced would be beneficial. Lower download times, energy consumption, and network congestion increase the network availability for other purposes.

Each of these components is the first proposed for its respective place in a P2P–MANET architecture. They do not rely on one another and the components may be used in whole or in part to construct a complete P2P–MANET system.

P2P computing is used for many purposes. In the performance evaluation sections of this thesis, we use file sharing overlays as our example overlay application type because they are the most popular P2P networks used today.

1.3 Thesis Organization

This thesis is organized as follows. Chapter 2 provides the background material and previous work necessary to understand the discussion that follows. Chapter 3 introduces the distributed bootstrapping scheme that allows nodes to join P2P–MANET

overlays. Chapter 4 presents the overlay topology control scheme that organizes the peers to minimize their cost. Chapter 5 discusses the incentive mechanism to encourage peers to contribute to the overlay and also the path selection algorithm that enables peers to select the lowest cost or lowest delay path. Chapter 6 presents the content distribution scheme that allows peers to download large files efficiently. Chapter 7 discusses the conclusions drawn from the thesis and discusses possible future research directions.

Chapter 2

Background

In this chapter we present some background material and previous work necessary to understand the proposed framework. The lifecycle of a peer in a P2P network consists of several steps. First, the node must locate the P2P overlay it wishes to join. Once it has done so, it must connect to at least one peer in the overlay. This process is known as bootstrapping and is discussed along with existing proposals in Section 2.1. Next, the peers must maintain logical connections between themselves. Which nodes should connect to each other relates to overlay topology control and is covered in Section 2.2. Once fully joined as an overlay member, peers will request and provide services from and to one another. For example, in a file–sharing network, peers will search for certain files and download them from other peers. This part of the P2P lifecycle incorporates many elements, including lookup, routing, and sometimes topology control and is discussed in Section 2.3. In resource–constrained environments such as MANETs, nodes must be convinced to contribute to the network. In the case of a file–sharing network, this involves sharing files as well as forwarding data when acting as an intermediate node on a multi–hop path. This may be accomplished

through the use of incentive schemes, some of which are discussed in Section 2.4. Once a peer has located an item of interest, it must select the best server(s) and path(s) and may try to download from multiple servers simultaneously. This is discussed in Section 2.5. Finally, once the server paths have been selected, it remains for the data to be transferred across the overlay. Section 2.6 presents some content distribution schemes. It also discusses network coding and how it can be leveraged to provide efficient content distribution. The final step occurs when the peer leaves the overlay network, which can be accomplished by simply sending disconnect messages to the peer's overlay neighbours. Section 2.7 provides an overview of our proposed P2P–MANET framework, and Section 2.8 provides a summary.

2.1 Bootstrapping P2P Overlays

Bootstrapping is the process of finding and joining a P2P overlay. Existing bootstrapping mechanisms are designed for wired P2P networks and generally expect the existence of a centralized infrastructure, such as a web server to facilitate the bootstrap process. Pure P2P systems cannot rely on any existing infrastructure or centralization. We now examine some existing bootstrapping schemes, first considering mechanisms that exist for wired P2P networks, then looking at how bootstrapping a particular structured overlay might work in a MANET.

Knoll *et al.* survey some bootstrapping protocols for wired P2P overlays [51]. The main techniques in use are divided into two categories, peer–based and mediator–based. The peer–based techniques include peer caching, in which each node keeps a list of previously active peers; multicasting, in which a single request is sent to multiple receivers; and random address probing, in which a random address is sent a

join request. Our proposed bootstrapping algorithm, presented in Chapter 3 makes use of multicasting and a form of peer caching. The second category consists of mediator–based schemes, including the use of a central server or host caches, which require the node to retrieve a list of peers from a specific URL; server lists, which contain peers that are usually online; and trackers, which form a central coordination point for peers. None of the mediator–based approaches are practical in MANETs because they all require some sort of centralized infrastructure, which cannot be assumed available in a MANET. Even distributed versions of these approaches may not work well. GWebCache is a web–based distributed host caching bootstrap system designed for Gnutella [37]. Khardari *et al.* [47] found that in fact, GWebCache behaves more like a centralized infrastructure complete with significant load imbalance. It was also found to misreport peer availability due to stale data and the absence of validity checks.

Conrad and Hof [22] propose a generic bootstrap service designed for the Internet. In their approach, all nodes belong to a specific bootstrap overlay, from which bootstrap information about a particular P2P overlay can be acquired. A similar idea, requiring all nodes to join a multicast group, is used in our proposed bootstrap algorithm.

The single, large P2P bootstrap overlay provides support for both large and small P2P networks, and is of particular advantage to smaller ones as it allows the use of Random Address Probing (RAP), which typically does not perform well with small overlay network sizes. RAP is a technique in which a node sends join messages to random addresses in the hope of contacting an overlay member. Conrad and Hof actually use a variation of RAP called Local RAP which checks IPs in the local

subnet first since they are more likely to be active. To distribute the P2P network information, any structured protocol can be used.

Castro *et al.* [18] similarly propose the idea of a universal overlay to provide the bootstrap service. Their design is based on the structured overlay Pastry [75] and provides mechanisms to advertise and discover services, to contact nodes, and also to acquire the application code required to install the desired service. They use multicasting, and provide persistent storage and distributed search.

Cramer and Fuhrman [24] examine how to bootstrap the structured overlay Chord [78] in an ad hoc network. They ignore node mobility and show theoretically that the process of spontaneously deploying Chord has a time complexity that grows linearly with network size. Simulation results further show that the bootstrapping procedure creates a significant traffic load imbalance at the join point and that as the network size increases it takes longer for the overlay to converge.

The process of bootstrapping a P2P overlay in a MANET must be fully distributed because there is no infrastructure. It cannot rely on "well–known" nodes either, since devices are mobile and energy constrained, so there is no guarantee that a node will remain for long in the network. Therefore, an alternative technique must be used. In Chapter 3, we propose a fully distributed bootstrapping mechanism that uses multicasting and response caching. This permits support for dynamic topologies and network participants.

2.2 Overlay Topology and Routing

P2P overlays may be separated into two classes: *structured* and *unstructured*. In a structured overlay, the topology is strictly controlled and data are kept with certain,

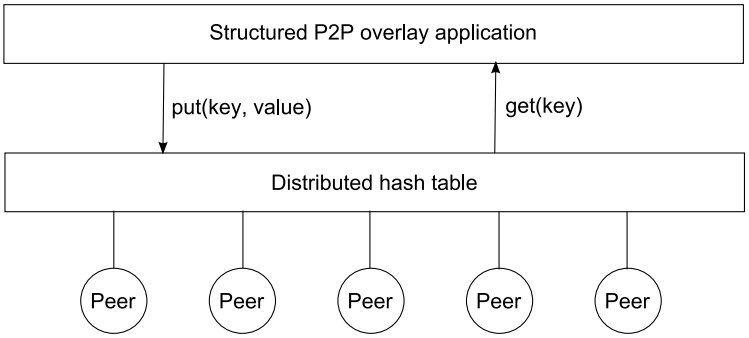


Figure 2.1: A structured DHT–based P2P overlay

specific peers, allowing for more efficient search and routing. Structured overlays use a Distributed Hash Table (DHT) which allows the location of data objects to be efficiently and deterministically found. Unstructured overlays do not implement a DHT or even require any specific topology. Peers join without knowledge of the topology, using relatively loose rules of connectivity.

Figure 2.1 shows how a DHT–based P2P overlay works. The P2P application exists over top of a distributed hash table layer. The interaction between them consists of 2 operations: *put* and *get*. Each node is associated with a NodeID and each data object is assigned a unique key, obtained by hashing the object. The identifier space of the NodeID and the key are usually the same, allowing keys to be mapped to active peers. The storage operation *put(key, value)* results in *value* being placed at the node that *key* maps to. The lookup operation *get(key)* results in the retrieval of *value* from the node that *key* maps to. Each peer "owns" a part of the space and stores the *(key, value)* pairs whose keys lie in its space. All of the *(key, value)* pairs form the DHT.

Peers maintain a routing table that consists of their neighbours' NodeIDs and IP addresses. Search queries are forwarded to neighbours with NodeIDs that are

"closer" to the key in the identifier space, with the definition of distance and the details of routing depending on the specific structured algorithm being used. DHT–based algorithms guarantee an upper bound on the number of overlay hops for lookup and routing peers, usually $O(\log N)$ for N peers. A downside to DHTs is that slight differences in data will generate different key values. For example, if there exist multiple copies of the same file with slightly different names, these copies would be mapped differently if the hash takes place over the file name. Also, keyword searching, wildcard, and boolean search operations cannot be supported in DHTs.

Due to the manner in which structured overlay algorithms construct the topology, the overlay network may not be efficiently constructed in terms of the number of underlay hops, resulting in highly variable performance. In addition, the tightly controlled topologies of structured overlays are typically undesirable in MANETs because node mobility would hamper their performance. There are many different structured overlays proposed, so we limit our discussion to three of the most well–known: Chord [78], Pastry [75], and CAN [74].

Chord [78] uses an m-bit identifier for NodeIDs and keys, with identifiers being ordered according to a circle modulo 2^m, called the Chord ring. The key k is assigned to the first peer whose identifier succeeds k on the ring. When a node joins the overlay, there will be some transfer of keys from its successor in the Chord ring. Similarly, when a node leaves the network, its keys are all assigned to its successor. Each peer maintains a routing table, called the finger table, with up to m entries. The ith entry of peer n's table contains the NodeID and IP address of the first peer that succeeds n by at least 2^{i-1} on the identifier ring. A Chord ring is shown in Figure 2.2. As an example, the fifth entry of Node 4's finger table corresponds to the first node greater

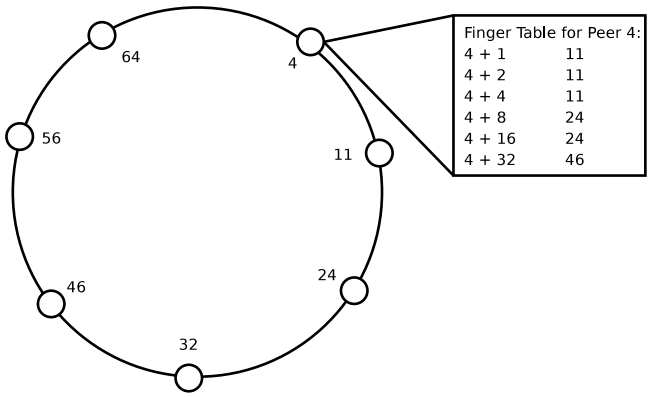


Figure 2.2: A Chord overlay with example finger table

than or equal to $4 + 2^{5-1} = 20$. Clearly, this is NodeID 24.

Pastry [75] uses a different routing technique, known as prefix routing. On joining the overlay, peers are assigned random 128–bit NodeIDs, which give a peer's position in the NodeID space ranging from 0 to $2^{128} - 1$. The NodeID space can more generally be thought of as a hypercube, which is a special instance of Plaxton–type networks. It is assumed that the NodeID assignments result in a uniform distribution of IDs throughout the ID space. Given a key, Pastry can route to the numerically closest peer in less than $log_B N$ steps, where NodeIDs and keys are considered as a sequence of digits of base $B = 2^b$. Peers forward messages to the neighbour whose NodeID has a prefix that is at least b bits longer than the one that the key shares with the current peer's NodeID.

Pastry peers' routing tables have $\log_B N$ rows, with each row holding $B-1$ entries. At row n, each of the $B-1$ entries refers to the peer whose NodeID shares the current peer's NodeID in the first n digits, but whose $(n+1)$th digit has one of the other $B-1$ possible values. These entries contain the IP address of a node with the required NodeID that is "close" according to a scalar proximity metric. When a peer joins the

overlay it contacts both the closest peer and the peer with the numerically closest NodeID. These two peers, as well as all nodes on the path between them, send their routing tables to the new peer, which uses this data to initialize its own routing table.

The Content Addressable Network (CAN) [74] uses a virtual d-dimensional Cartesian coordinate space on a d–torus partitioned among peers. Each peer has a distinct zone within the space. Peers maintain a routing table with the IP address and virtual coordinate zone of each neighbour in the space. Messages are routing using a greedy algorithm that forwards to the peer that is closest to the destination coordinates. The routing performance is $O(d \times N^{\frac{1}{d}})$. Key k is mapped to a point P in the coordinate space using a uniform hash function. When a peer joins the overlay it randomly chooses a point in the coordinate space, and the peer responsible for that zone splits it into two, with half being the responsibility of the new peer, and the other half the old peer. Figure 2.3 shows an example of a 2–dimensional CAN overlay. Each node is responsible for one of the zones. For clarity, the torus shape of the overlay is not shown, but it must be noted that the space wraps on the edges.

In unstructured overlays, search queries are flooded to peers, generally with a limited scope to prevent overloading the network. Flooding–based techniques work well in locating readily available objects and are resilient to churn, however they may not locate rare objects that exist in the network. They also face difficulty scaling, as is typical of flooding-based techniques. Despite these disadvantages, unstructured overlays are more commonly used today. They also provide support for keyword, wildcard, and boolean search operations. Hora *et al.* [26] have found that in MANETs, unstructured protocols are more resilient than structured ones at the cost of higher energy and delay. There are many different unstructured overlays, so we limit our

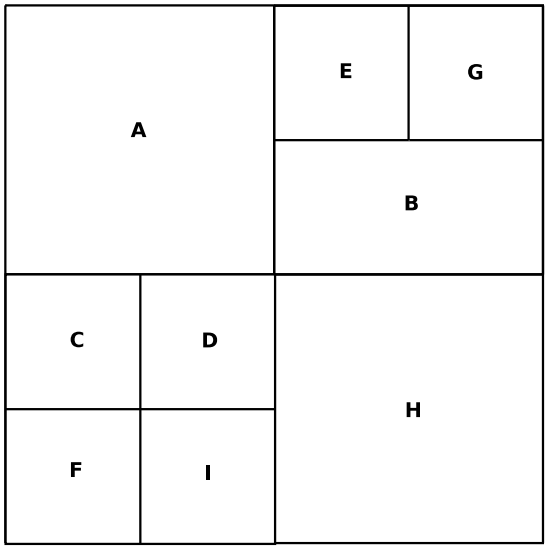


Figure 2.3: A 2–dimensional CAN overlay

discussion to three of the most popular: Gnutella [37], FastTrack [7], and BitTorrent [11].

The original Gnutella, known as v0.4 [37], allows users to connect to one another with loose rules. There is no precise control over the topology or file placement. To locate data, peers query their neighbours, who then pass this query to their neighbours. The query is thus flooded to all peers up to a certain radius. The design has proven very resilient to peers entering and leaving, but the search mechanism is not scalable and can create a high load on the network. When a peer joins the overlay, it initially connects to a well–known host. Gnutella–like, unstructured topology overlays are a good choice for MANETs since they are able to handle churn and node mobility fairly well. Newer versions of Gnutella use the idea of ultra–peers, similar to the technique used in the FastTrack network described next, in order to achieve greater performance and scalability. A Gnutella v0.4 overlay is shown in Figure 2.4.

FastTrack [7] is a more structured overlay than the original Gnutella and was

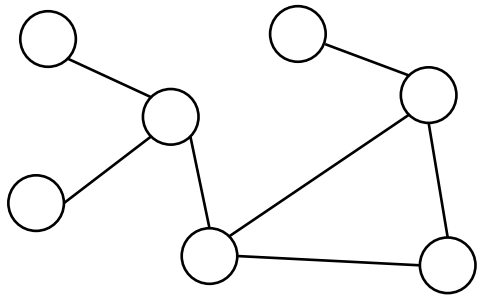


Figure 2.4: An unstructured Gnutella overlay

used in the once–popular Kazaa file–sharing software, as well as by the still popular Voice–over-IP application Skype. Ordinary nodes connect to special nodes, called super–peers (or ultra–peers in Gnutella parlance). The super–peers are selected due to their higher bandwidth, disk space, and processing power, and they are used to facilitate search by caching meta–data. Ordinary peers transmit the meta–data of files they are sharing to their super–peer. File queries are also sent to super–peers. Super–peers broadcast search requests to one another, but due to the pruned overlay, it does not result in overwhelming of the network. The exact working of the super–peer topology is unknown due to its proprietary nature. In a MANET, the concept of super–peers would result in some peers using their energy at a much higher rate than others. Therefore, it would be necessary to cycle this responsibility among nodes in a MANET if a FastTrack–like overlay is used. An example FastTrack overlay is shown in Figure 2.5.

BitTorrent [11] focuses on the file download aspect of a file–sharing P2P network and excludes any search functionality. A centralized node, known as the tracker, is used to coordinate nodes. Peers are given the IP addresses of several other peers and attempt to maintain connections with them. They announce to their neighbours which pieces, or chunks, of the file they have, and download requests are fulfilled

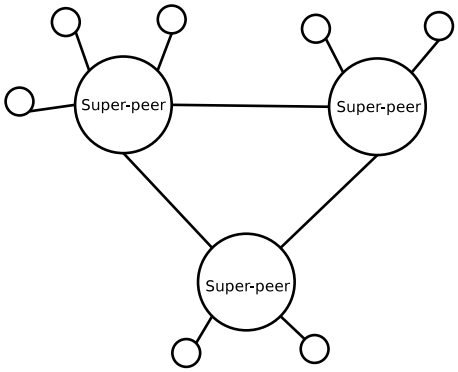


Figure 2.5: A FastTrack overlay

using the *tit–for–tat* incentive mechanism. That is, if a neighbour uploaded a chunk in the previous round, then its next request for a chunk will be accepted in the current round. If the node ignored a request in the previous round, its request in the current round will also be ignored. While maintaining their connections, peers occasionally randomly connect to new peer in an attempt to find a better neighbour. BitTorrent does not maintain any regular topology; peers connect to the tracker and to their neighbours. BitTorrent is not a good overlay model in MANETs due to the use of a centralized tracker, and also the lack of a query mechanism.

In constructing an overlay topology for P2P–MANETs, it is important to consider the topology of the underlying physical links. Liu [60] showed that in an unstructured overlay, more than 70 percent of the links do not reflect the underlay topology due to the selection of random neighbours. Tamura *et al.* [79] show that even in wired networks, end–to–end file transfer delay is smaller when the overlay and underlay topologies are the same. Unnecessary energy consumption will also occur when MANET nodes are involved in routing overlay traffic due to a mismatch in the overlay and underlay topologies.

Fabrikant *et al.* [31] introduce a game–theoretic approach to the construction of

topologies. The advantage of this approach is that selfish node–agents, working in their own self–interest, and without any centralized design or coordination, are able to determine which links to connect between themselves. The networks are intended for peering agreements between autonomous systems on the Internet, not as P2P overlay networks, but the approach is general enough to have inspired our overlay topology control algorithm, as discussed in Chapter 4.

The network creation game, as it is dubbed, requires nodes to pay for the links that they establish to others while also considering the distance to all destinations. The cost to establish a link is constant, the distance is the hop count, and agents try to minimize their total cost. A single parameter, α, is used as a measure of the tradeoff between establishing more links and decreasing the distance to other nodes. Therefore, it is in an agent's interest to have fewer direct neighbours, while keeping the entire network as close in distance as possible. The authors show that determining the Nash equilibrium, the stable topology, is an NP–hard problem. The "price of anarchy", a measure of the difference between the socially optimal topology and the topology attained when agents act in their own self–interest, is also discussed.

Chun *et al.* [21] extend the work of Fabrikant *et al.* by simulating the network creation game under various network sizes and configurations. They show that without constraining the node degree, resulting topologies are near–star configurations, in which a small number of nodes will maintain a large number of connections. This allows other nodes to "free ride" off of those nodes by connecting to them and achieving both a small number of connections and also a low distance. This results in less resilient topologies because a small number of node failures will result in a large degree of disconnection. Therefore, the authors conclude that there is a fundamental

tradeoff between performance and resilience.

Moscibroda *et al.* [67] also extend the work of Fabrikant *et al.*, examining the effect of the network creation game on P2P topologies. The latency between nodes is used as the distance metric, and the authors provide an upper and lower bound on the price of anarchy. They go on to prove that there exist metric spaces for which no Nash equilibrium exists, meaning that even in the absence of peer churn, the topology will never stabilize. Our results appear to confirm this for P2P–MANET networks as well.

Constructing and maintaining the P2P overlay topology in a MANET must be fully distributed due to the absence of infrastructure. It must support node mobility and consider energy consumption. Chapter 4 presents both a fully distributed overlay topology control algorithm based on the network creation game, and an associated heuristic algorithm that supports the dynamism of MANETs and considers neighbours' energy levels.

Table 2.1 provides a comparison of the algorithms discussed. Since the FastTrack network is proprietary, the exact nature of the connections between super–peers is unknown. The Gnutella network has a routing distance that may be the diameter of the network (D) since peers join randomly. There is no exchange of information on joining, so the state size and join complexity are constant. In the BitTorrent network, peers connect to the tracker, and to the fixed number of peers they are exchanging data with, so it maintains a constant state size and join complexity. This also causes its irregular shape, and since it provides no lookup function, the routing distance comparison does not apply. Many of the comparison factors for the network creation game cannot be determined *a priori* since they depend on the

Table 2.1: Comparison of overlay topology algorithms

	Type	Overlay shape	Routing distance	Table size	Join complexity
Chord [78]	Structured	Ring	$O(\log N)$	$O(\log N)$	$O(\log^2 N)$
Pastry [75]	Structured	Plaxton–type	$O(\log N)$	$O(\log N)$	$O(\log N)$
CAN [74]	Structured	d–dimension torus	$\frac{1}{4}d\sqrt[d]{N}$	$2d$	$O(\log N)$
Gnutella [37]	Unstructured	Random	$O(D)$	$O(1)$	$O(1)$
FastTrack super–peers [7]	Unstructured	Hierarchical	Unknown	Unknown	Unknown
BitTorrent [11]	Unstructured	Irregular	N/A	$O(1)$	$O(1)$
Network creation game [31]	Unstructured	Indeterminate	Indeterminate	Indeterminate	NP–hard

specific underlay topology and the parameter α.

2.3 Lookup and Routing in MANETs

Most attempts to combine P2P networks and MANETs have used cross–layer techniques to merge existing structured algorithms with MANET routing protocols. They focus on improving the lookup and routing performance in MANETs by exploiting P2P DHT techniques. In doing so, they often force a specific routing protocol and overlay topology to be used, they focus on node mobility only tangentially, and they often ignore consideration of energy consumption. These factors prevent general–purpose P2P overlay networks from flourishing in MANETs. What follows is a discussion of some of these schemes.

Delmastro [27] examined the performance of a Pastry–like algorithm over a MANET using the well–known Ad hoc On–Demand Distance Vector (AODV) routing protocol [71] and determined that the performance was poor due to the heavy overhead needed to maintain the large number of connections. It was recommended that the overlay and underlay networks should more closely correspond in order to improve performance.

Cramer and Fuhrmann [25] simulated the performance of Chord in a MANET and found that it too performed poorly. They determined that this occurred not because of the overhead of maintaining the DHT, but because of its pessimistic timeout and failover strategy. When a node moves or a packet is lost, Chord assumes that the node has left the network and removes its state information from the overlay, leading to incorrect behaviour for lookups.

Hu *et al.* [44] aim to provide a way for MANETs to scale to larger numbers of nodes and attempt to use a P2P protocol to do so. They propose a combination of the Dynamic Source Routing (DSR) [46] MANET routing protocol and Pastry, to make DPSR (Dynamic P2P Source Routing). Nodes are assigned a NodeId that is the SHA-1 hash of their IP address and messages are routed based on the hash of the destination address using Pastry's prefix–based routing. Each Pastry hop is actually a multi–hop source route and routes to NodeIds are stored in the routing table as source routes. Due to the use of Pastry, only $O(\log N)$ routing table entries are needed. There are no simulation or performance results to demonstrate the feasibility of DPSR.

Zahn *et al.* [89, 85] adapt the Pastry algorithm for MANETs by considering the MANET topology while constructing the DHT substrate. Pastry is combined with AODV at the network routing level. MANET nodes are divided into clusters, and in each cluster a random landmark ID is chosen. The IDs are evenly spread throughout the address space and once selected are broadcast throughout the cluster. Each node then chooses a random ID based on the landmark ID so that it is easy to distinguish which cluster a node belongs to. The closest landmark by hop count is chosen when a node joins a cluster. If another landmark is determined to be closer, the node

joins that cluster and resigns from the old one. To forward traffic, the data is first sent to the correct cluster, based on the landmark ID. Once within the cluster, it is forwarded to the appropriate node based on the NodeID. Pastry maintains a large routing table, the upkeep of which would overwhelm a wireless network. Therefore, the authors propose to instead only track landmark keys, increasing the $O(\log N)$ search bound of Pastry. This approach enforces a loose structure on the overlay and focuses only on the lookup function. It does not consider energy consumption explicitly, and requires that the MANET use the AODV routing protocol.

Klemm *et al.* [50] propose an algorithm called Optimized Routing Independent Overlay Network (ORION) that merges AODV with many P2P overlay functions to create a file–sharing network. Overlay connections are set up based on file queries and transfers, which matches the topology of the underlying network, combining query processing with network layer route discovery. When a client issues a query, it is broadcasted to all nodes and those nodes with matching files respond back. The results are aggregated such that redundant responses are removed by upstream nodes on the return path back to the client. This information is stored as an alternative path by the intermediate node and used in the case of path failure. Nodes maintain a file routing table as well as a response routing table. The file routing table indicates the next hop to a file. One problem with ORION is that intermediate nodes should not be making the decision for the client as to which server is best for it, as is done with the route aggregation technique. Allowing the client to receive multiple responses not only enables it to select the best server, but also to download the file from multiple servers simultaneously.

Schollmeier *et al.* [76, 39] present the Mobile Peer–to–Peer Service (MPP), which

extends DSR in cross–layer fashion to allow locating of nodes by means other than IP address. This enables support of peer location at the overlay level. File search queries are flooded to all nodes, similar to a DSR route request query. File response queries are responded to just like DSR's route reply messages, including the complete path between source and destination. Simulation results show that MPP outperforms ORION.

Recently, Banerjee and King [8] have proposed algorithms to construct ring–like overlays in MANETs in a distributed manner. Their motivation is the establishment of control operations such as mutual exclusion and clock synchronization, but it may be useful for the implementation of ring–like overlays such as Chord on MANETs.

These efforts focus on improving lookup and routing performance in the MANET, while ignoring other aspects of P2P–MANETs, such as bootstrapping and content distribution. Therefore, they do not provide a complete, practical P2P–MANET system, which is the goal of this thesis.

2.4 Incentive Mechanisms

A successful P2P overlay requires users to contribute resources, such as disk space, bandwidth, or computation. Most existing P2P networks suffer from large numbers of freeloaders, nodes which only take from the network and give nothing back [1]. Therefore, it is important to convince users to share their resources through the use of an incentive scheme [55]. Incentives schemes exist in the literature for both MANETs and P2P networks, but separately. We now examine incentive schemes for MANETs, followed by incentive schemes for P2P networks.

2.4.1 Mobile Ad Hoc Network Incentive Schemes

There are two basic classes of incentive mechanisms in mobile ad hoc networks: reputation–based and credit–based. The general idea of reputation-based systems is that nodes earn a reputation based on past behaviour. When they forward packets their reputation increases. When they don't forward packets, their reputation decreases. Nodes with a low reputation are avoided in path selection, and may be punished. The basic idea in credit-based schemes is that nodes earn some form of credit when they forward packets and spend credit to send their own data.

Reputation–Based Schemes

Nodes monitor their neighbours and alert others when they feel a node is not cooperating [61]. Misbehaving nodes are avoided when selecting paths. Other schemes attempt to punish misbehaving nodes by not forwarding their traffic [13, 63]. These schemes require observations to periodically be sent network-wide, which may create excessive network traffic and increase energy expenditure.

Conti et al. [23] propose that nodes maintain a reputation table for their neighbours but not share it with anyone. The reputation is increased when the node receives an ACK for sent packets and decreased otherwise. If no ACK is received it is uncertain which node along the path did not forward it, but the neighbour's reputation is still decreased because that path proved itself untrustworthy. First-hop selection is based on the neighbour with the highest reputation. Intermediate nodes cannot determine reputations, only the source node can.

OCEAN [9] assumes that antennae are omni-directional and listens for neighbours to send a packet. Sent packets are verified based on the checksum and the node's

reputation increases if it is correct, or decreases if not. This scheme allows intermediate nodes on the path to determine reputations, but requires nodes to listen to all traffic, which has an undesirably high energy usage.

Credit–Based Schemes

Buttyán and Hubaux [15] propose a simple scheme in which a node uses one credit for each packet sent and receives one for each packet forwarded. A tamper–resistant module in each device ensures that credit is not forged. The greatest weakness of this scheme is the tamper–resistant module itself, which requires either a hardware module, or a cryptographic scheme.

Other schemes such as Sprite [90] and PIFA [88], require nodes to maintain receipts of packets sent and received and periodically send these to a centralized server which determines the amount of credit owed and deserved. Centralized servers are typically an impractical requirement in MANETs.

Ad hoc-VCG [6] integrates with the underlying routing protocol. The network graph is weighted on the energy cost of sending information. The sender chooses the lowest cost route, but in order to disincentivize nodes from lying about their energy cost, it pays a premium that is above the lowest route cost, but below the next lowest route cost.

In a scheme called FAIR [65], the cost of forwarding is not fixed. Instead, it is a function of the current node's CPU, bandwidth, battery, external demand, and estimated cost of forwarding to the next hop. The sender estimates how much the next-hop will charge it, and is then billed afterward based on the calculation. The next hop is then in charge of ensuring the packet reaches the destination. In a

sense, the intermediate nodes "buy" the packet from upstream nodes and "sell" it to downstream nodes. A drawback of this scheme is that because nodes only estimate the cost, in order that they not be too far off, a node can only change its price gradually. This may not suffice in dynamic MANETs.

Qian et al. [73] argue that credit schemes are unfair to nodes on the boundary because they do not get as much opportunity to forward traffic. Their scheme proposes charging based on how much the sender can afford (i.e. based on how many credits it currently has). This way boundary nodes have a better chance of being able to send their data, and central nodes will use up their credits more quickly and so must continue forwarding data. The drawback to this idea is that it does not allow nodes to build up credit when they have little data to send, and thus effectively punishes them for not injecting traffic into the network.

2.4.2 Peer–to–Peer Network Incentive Systems

Most peer–to–peer network incentive systems can also be divided into reputation– or credit–based. However there are some schemes which do not neatly fall into either of these two categories.

Reputation–Based Schemes

Papaioannou and Stamoulis [69] propose a system in which peers rate one another based on their past transactions. Both parties get to vote, and if they disagree they are both punished since one must be lying. The degree of punishment is determined by a "non-credibility" metric that is updated based on past behaviour. To implement the scheme requires a PKI and encryption, which may not be feasible in MANETs.

The non-credibility measure and ratings are sent to a group of peers that track the information for a certain node. These peers are determined via a DHT.

Buchegger and Le Boudec [14] propose a fully distributed reputation system in which peers maintain both a reputation rating and a trust rating about all other peers they are interested in. The idea of maintaining two ratings is to prevent the spread of false reputation data while avoiding the situation of relying on only the peer's direct observations. Periodically nodes exchange first–hand reputation information and use a modified Bayesian approach to merge only compatible ratings. The trust rating is modified based on how compatible the second–hand information of a peer is, based on past reputation ratings. Despotovic and Aberer [28] use a similar idea, but instead of a Bayesian approach, they advocate the probabilistic technique of maximum likelihood estimation, which they say requires fewer reports from peers.

Dutta *et al.* [30] propose a scheme in which users maintain a sliding window rating for all other users they have interacted with. When a node wishes to determine the reputation of another node, it proceeds differently based on the type of network. In a structured network, like Chord, a supervisor is selected for each node and maintains the rating of it. The peer contacts a node's supervisor to determine its rating, and submits a rating after its transaction is completed. In an unstructured network, like Gnutella, nodes keep a list of peers whom they have interacted with, and a peer can sample these to get a rating.

Grolimund *et al.* [38] state that when a node wants to determine a peer's reputation, it should not need to consult other nodes. Therefore they propose a system named Havellar, in which a node keeps track of reputations for all the peers it interacts with. Every round, it sends this table to k successor nodes. These nodes are

determined by performing k hash functions on the node's address (i.e. the successors are always the same). The successors merge their observations with that of the predecessor. The authors add a second round of succession to cope with large networks. The system was designed for a P2P file storage system where nodes are up for hours every day and requires a lot of messages for maintenance.

Credit–Based Schemes

Liao et al. [59] propose a very simple idea for incentives: each byte uploaded gives a certain number of credits, each byte downloaded uses a certain number of credits, and each second online gives a certain number of credits (awarded at $\frac{1}{10}$ the value of downloading). This encourages users to share files, and also to stay online.

Vishnumurthy et al. [83] also propose a fairly simple scheme in which a root node maintains the locations of all files. A peer contacts the root node, which in turn contacts all servers with the file, who can then contact the peer with their price. The peer chooses the lowest priced server. Credits, referred to as karma, are maintained in a "bank–set" of nodes for each peer so that they cannot be tampered with.

Kung and Wu [54] state that a node should only provide just enough service that it can use all of its credits. When a node provides a service, it gets 2 credits. When it transports data, it gets 1 credit. These credits are placed into a service matrix and from this a usage matrix and eigenvector-based rankings on service and usage reputation are computed. When node X wants a service from node Y, Y will agree or not based on how close its service ranking is to what it is desired to be. If Y agrees, it will contact a few different nodes to see if X's service rating is high enough and usage rating low enough. It contacts 2 "benchmark nodes", one with a high service rating

and one with a high usage rating, and also some other nodes to sample the values for the matrix. Based on X's comparison with these nodes, Y will agree or not to serve X.

Yang et al. [87] consider a P2P environment in which the search query itself is bought by service providers, in the hope that the client will buy the actual service from them. In essence, the server pays for the right to sell its service. Before the query is sent, a partial "Right To Respond" (RTR) containing the reputation of the querier, the query itself, and the price of the RTR are sent to peers. If the peer agrees, the full RTR is then sent to it, which includes the identity of the querier. This way it cannot cheat the system by not first buying the RTR. A server that acquires the RTR can then respond to it with the possibility that the client will buy its service. It can then also sell the RTR to other peers to recoup some of its costs. There can even be "wholesaler" peers who just buy and sell RTRs without actually providing any service. This system encourages propagation of the query, and ultimately the client may get a good set of responses.

Gupta and Somani [40] propose a strategy for Chord–based networks in which the client determines the utility of the requested object and from that calculates a maximum price it is willing to pay. It also calculates a "marginal cost" (MC), which accounts for bandwidth, energy, CPU, etc. These values are sent to "terminal nodes", which index the object locations. In parallel, each of these terminal nodes adds their MC to the total and then passes it along to its successor, which adds on its MC, until it reaches the server. The server then gets n parallel requests from the same client. It looks at the highest price, which is the maximum price minus total MC, and that chain of nodes is selected. The actual price paid however is the second highest price

(this is the procedure for a Vickrey auction, which is designed to maximize public good, not self–interest). Each node in the winning route gets the profits in proportion to their MC.

Wongrujira et al. [86] propose a system for Chord. The query is split into 2 and sent in both directions along the ring to prevent one side from cheating, since the message is useless without both parts. Along the way to the server, each node adds its forwarding cost to the query. The client can then choose the cheaper side to send the data on.

Alternative Schemes

Anagnostakis and Greenwald [5] focus on an exchange or barter economy. They believe that peers prefer to trade with someone who has something they want. The exchange can even be n-way, in a logical ring. If there is excess capacity, then a peer can send to another who doesn't have anything they want. The authors claim that finding all n nodes to exchange with is not hard. Peers simply examine their incoming request queue and see if a node on there has something they want. This is a 2-way exchange. If we make a directed graph with this dependency as an edge and the peers as vertices and look for a cycle of length n, then that is an n-way exchange. The authors say that they have empirically determined that there is not a substantial improvement in the likelihood of an exchange for $n > 5$, so $n = 5$ is sufficient to look for.

Moreton and Twigg [66] generalize and combine the ideas of reputation and credit systems into a "stamp trading" system. In this system, a node can issue as many of its own branded stamps as it pleases, which can later be redeemed. Other nodes

can also trade these stamps, and do so based on its exchange rate. If node A does not provide good service, the value of its stamps goes down and when it later wants a service, its stamps may be devalued to the point that nobody will exchange them. While a novel idea, a large drawback to this system is maintaining the exchange–rates, which would need to be centralized or it require a lot of overhead to maintain in a distributed fashion.

Only one incentive scheme for P2P overlays in MANETs could be found. Zhu and Mutka [91] propose a scheme to allow MANET users to share access to Internet services. In the scheme, named CHUM, users maintain trust values for one another which combine their own observations with recommendations from others. On top of this trust scheme exists a credit scheme, in which nodes can borrow from one another and obtain larger credit limits if they are good based on their trust values. A downside to the scheme is that nodes must maintain trust histories and credit limits for all other peers.

For a P2P–MANET, it is essential that nodes be rewarded for both sharing data and providing services, and also for forwarding data on behalf of other MANET users. It is also important to reward MANET nodes that are not currently participating in the overlay so that if they do join in the future, they may use the goodwill they have earned. The incentive scheme proposed in Chapter 5 satisfies these goals.

2.5 Server Selection

Selecting the optimum server peer for a file download is examined by Adler *et al.* [2, 3]. The pricing mechanism and how client nodes determine what peers have what files and what they are charging is beyond the scope of the problem considered. It is assumed

that the servers have somehow indicated to the client the cost of downloading the file from them as well as their available uplink bandwidth. The cost may be either real money or some form of virtual currency.

Given a set of servers, their upload bandwidth, and their prices, the client must select how much to download from each server in order to minimize the download time, while also not exceeding a set budget. The optimum solution is given in the form of a linear program. The goal of the program is to minimize the maximum download time from each server. Over all servers chosen, the slowest one, i.e., the one with the maximum upload time, determines the total download time, so this is the time the program must minimize. The constraints are that the cost must be less than the budget, the bytes downloaded over all servers must obtain the complete file, and the bytes per server must be greater than zero.

Next, the authors examine a similar problem as it applies to real–time streaming. The problem is now to select peers such that the total streaming cost is minimized while ensuring that at any point in time, all bytes before that point in the stream have already been downloaded. This problem has the added challenge of scheduling the parts of the file correctly. They propose two approaches, time segmentation and rate segmentation. In time segmentation, the video is split up by time and each server streams only the segment it is responsible for, while the client buffers. The problem with this approach is that the total amount being sent to the client may be more than it can handle. In rate segmentation, each server sends a portion of the frame and the total amount uploaded to the client is the playback rate. The authors then propose a linear program for rate segmentation, which attempts to minimize the cost of the rates across all servers such that the bit rate is greater than the playback rate.

Han and Xia [42] study techniques for server selection in an overlay network structured as a hypercube. The client selects k server candidates, where k is the degree of parallelism, to satisfy a cost–minimizing criterion that reflects either network resource usage or interference among the connections from the servers to the client. The interference is minimized by selecting those servers whose position within the hypercube does not cause traffic collisions on the path between the server and the client. This can be determined based on the node IDs, which are indicative of how they are connected within the hypercube. The network resource usage is defined by the number of flows over a link, with the goal of avoiding those that are most used. The minimum interference algorithm runs in $O(n \log n)$ time and the resource usage algorithm runs in $O(nm)$ time, where n is the number of possible servers, and 2^m is the size of the network.

In a P2P–MANET, it is important to consider the costs of all intermediate nodes on the path to the server as well as the server itself. This allows clients to download via multiple paths, and even use the same server for some of those paths. Also, because most P2P–MANETs are likely not structured overlays, the server selection algorithm should support unstructured networks. The path selection algorithm presented in Chapter 5 satisfies these goals.

2.6 Content Distribution

Content distribution involves transferring data, or content, from a set of source nodes to a set of destination nodes. Its definition often includes searching for the content as well, but in this thesis, we do not include this aspect in our definition of content distribution, and instead focus only on the content transfer component.

In the simplest content distribution system, a client contacts a server and downloads the content from it. However, as the number of clients grows, the server's uplink bandwidth becomes a bottleneck and better techniques become desirable. Taking advantage of the uplink bandwidth of the clients that are downloading is a common way to more quickly distribute content.

One approach is to use application–layer multicasting and allow all nodes to contribute to the distribution by uploading the parts of the file they possess [45]. The difficulty with this approach is the construction of the multicast tree, and whether to use source–based trees or a shared–tree.

Another approach is to forego multicasting and instead allow nodes to connect directly to one another and send content, as in a P2P overlay. The most popular such approach is BitTorrent [11]. A downside to this approach is that the overlay network does not closely reflect the underlay, resulting in inefficient use of network links.

Forward Error Correction based on erasure codes can also be employed [17]. In this approach, sometimes referred to as a "digital fountain", the content is split into k equal length packets. The original data is encoded by a server, which streams distinct, encoded packets. Clients accept these packets and distribute them amongst themselves, until they have received k distinct packets. In the ideal situation, the client can reconstruct the data after receiving k packets. In practice, however, due to the existence of cycles in the topology, clients tend to receive duplicate packets, and locating the missing data packets is challenging. Some approaches have been proposed to overcome this problem [53, 16], but they involve additional complications such as bloom filters or gossip packets.

A generalization of the digital fountain idea involves the use of network coding.

With this approach, both servers and clients encode data, which mitigates the problem of clients sending packets that may not be of use to one another, as in the digital fountain model. Codes are combined with one another by clients, which reduces the chance of a client receiving redundant data.

Network coding was introduced by Ahlswede *et al.* [4] and linear network coding was introduced by Li *et al.* [58] as a technique to save bandwidth. There are several tutorial papers covering practical network coding [20, 33, 19]. Network coding is a form of information spreading in which nodes, instead of simply forwarding data packets, combine, or encode, several packets together using the XOR operation. The idea is to shift some of the work burden from the network to the nodes' computational abilities. This is a good tradeoff since network bandwidth is generally more expensive, i.e., slower, than computation. One major benefit of network coding is that encoded packets can be further encoded. This allows nodes to encode data without first decoding it, so that they can encode packets for which they do not yet have the completed data.

The original packets are associated with a set of coefficients over the field. These coefficients are multiplied by the original packet, s bits at a time, and XORed with the same bit positions of the other packets to be combined. For example, given two original messages, M_1 and M_2, and two coefficients g_1 and g_2, the encoded message would be g_1M_1 XOR g_2M_2. The linear combination of the data, interpreted as a set of numbers over a finite field, is then transmitted in place of the original packets. The data of size s is considered as a symbol over the field $\mathbb{F}_{2^s}$. Therefore, a packet of length L consists of L/s symbols. This encoded data, known as the information vector, is sent along with the set of coefficients, known as the encoding vector. The

encoding vector is needed by the receiver to decode the data.

The coefficients are selected at random, in a completely independent and decentralized manner. It is believed that even with a small field size, such as 2^8, the probability of selecting linearly dependent combinations is negligible [33].

With network coding, nodes receiving the encoded packets do not need to worry about obtaining specific packets. Instead, they must obtain a sufficient number of linearly independent, or "innovative", packets. The encoded packets, along with the given encoding vectors, are considered as a system of equations, where the original messages are the unknowns. For a set of n original messages and m received messages, we therefore have m equations with n unknowns, and so we must have $m \geq n$ to solve the system. Furthermore, there must be at least n linearly independent equations. Any known technique, such as Gaussian elimination, may be used to solve the system and thus recover the original data.

Katti *et al.* present the first implementation of practical network coding to the wireless environment [48, 49]. The approach, called COPE, is for nodes to listen and remember what packets neighbors have received so that the "best" source packets are encoded to minimize the number of transmissions. This opportunistic approach relies only on local information and exploits coding opportunities in real-time. All nodes in the wireless network participate. Transmitted packets are annotated to inform neighbors which packets the transmitter has heard. When a node sends data, it uses the knowledge of what its neighbors have received to perform opportunistic coding. Simulation results as well as multi-node test beds confirm the benefits of network coding.

Some recent work has examined the use of network coding in P2P file sharing over

wireless mesh networks using the technique of opportunistic coding [41, 29]. However these works assume that all nodes in the network are also part of the P2P overlay, an assumption which is generally not the case.

Gkantsidis *et al.* bring practical network coding to the realm of P2P file sharing [34, 35, 36]. Their system, dubbed Avalanche, is designed to allow nodes to more quickly download a large file. The source node along with other peers that have parts of the file, encode all the blocks they have available. As long as a downloading peer gets enough linearly independent blocks, it can decode them to retrieve the entire file. The authors show that even if the server leaves shortly after seeding one copy, the network will still be able to get a high completion rate for the file. They also show that the CPU and I/O requirements of encoding and decoding add minimal overhead. Avalanche assumes the use of a centralized tracker which coordinates the nodes. In a MANET, such a node is impractical. Furthermore, Avalanche is designed for wired networks and does not consider energy use.

Small *et al.* examine the use of network coding in a P2P network [77]. They conclude that peers are not very likely to receive independent blocks unless the block comes from a peer that has fully decoded the blocks first, meaning it has already received the entire file. Dependent blocks are a waste of bandwidth. The authors also show that the topology of the network has an effect on the usefulness of coding. Dependent blocks are received by nodes due to a common upstream parent and are a function of the "aggressiveness" of nodes. That is, the more frequently nodes send blocks to one another, the more likely they are to be dependent. However, decreasing the frequency increases delay. The scheme proposed in Chapter 6 allows nodes to select which peers to download from, and how much to download from them, reducing

the chances of receiving dependent blocks.

2.7 P2P–MANET Framework

To realize the benefits of a P2P file–sharing network while addressing the issues of running the overlay in a MANET, we propose a P2P–MANET framework which consists of five main components: a bootstrapping mechanism, an overlay topology control algorithm, an incentive scheme that employs a pricing method, a server path selection algorithm, and an efficient file transfer method. The framework aims to achieve low delay and energy consumption, while supporting peer mobility. It is our objective to achieve superior performance in each of the components, as compared to existing solutions. Our solution is specifically designed with MANETs in mind, while existing solutions are not. The bootstrap mechanism is completely decentralized, the overlay topology control algorithm considers the physical topology, the incentive mechanism encourages users to contribute to the P2P network, the download path selection algorithm chooses the best way to download a file from multiple servers in the fastest time subject to a budget constraint, or vice versa, and the efficient file transfer method supports fast file downloads with low energy consumption. Each of the proposed components is independent of the others and may be used together or combined with other approaches. The components are examined in detail in the remainder of this thesis.

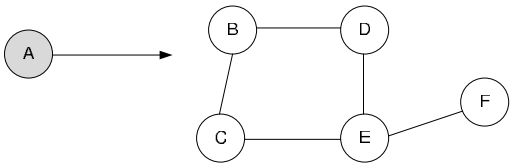
Figure 2.6 illustrates the function of each of the framework components. Figure 2.6a shows an overlay, in which a new node, A, wishes to join. This is accomplished with the bootstrapping component. Figure 2.6b shows that node A has successfully joined the overlay. It is the job of the topology control algorithm to determine the

set of links between peers. In Figure 2.6c, node A is searching for some content. The credit–based incentive component ensures that peers E and F, who are willing to upload the file, are paid an amount equal to the first number shown next to them. The second number reflects the estimated delay to serve the file. The intermediate nodes, B, C, and D, may also charge. For clarity, this is not shown in the figure. Figure 2.6d shows the path that A has chosen to download the file. It may download some parts of the file from E, and the remaining parts from F. Finally, Figure 2.6e shows all peers involved in the content distribution of another, larger file. The arrows show the direction of content flow. Some peers are only uploading to others, some are only downloading, and others are doing both.

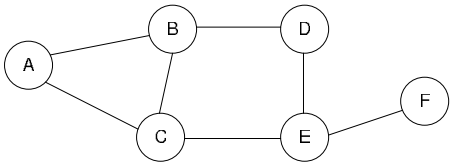
The framework presented in this thesis does not rely on any particular MANET routing protocol for either unicast or multicast routing. However, since a multicast and unicast routing protocol must be selected for the simulations, Appendix A provides a discussion of the AODV, MAODV, and DSR routing protocols.

2.8 Summary

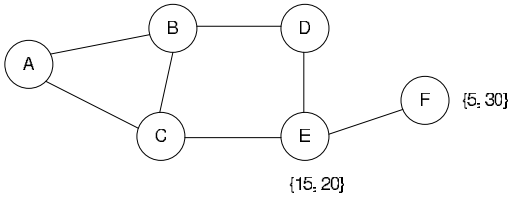
In this chapter we reviewed the main components of our proposed P2P–MANET framework: bootstrapping, overlay topology control, incentive mechanisms, server path selection, and efficient content distribution. Related works in the literature were discussed. In this thesis, we propose algorithms and heuristics for each of the previously mentioned components that work independently of one another. Though they are capable of operating independently, when combined together, they allow a practical P2P–MANET system to be built.



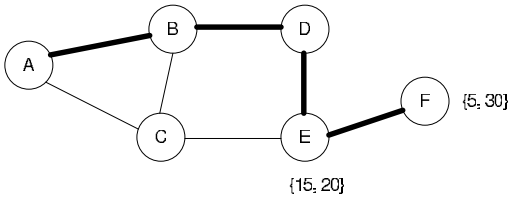
(a) Bootstrapping



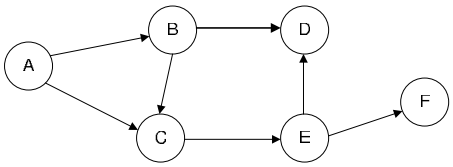
(b) Topology control



(c) Incentives



(d) Path selection



(e) Content distribution

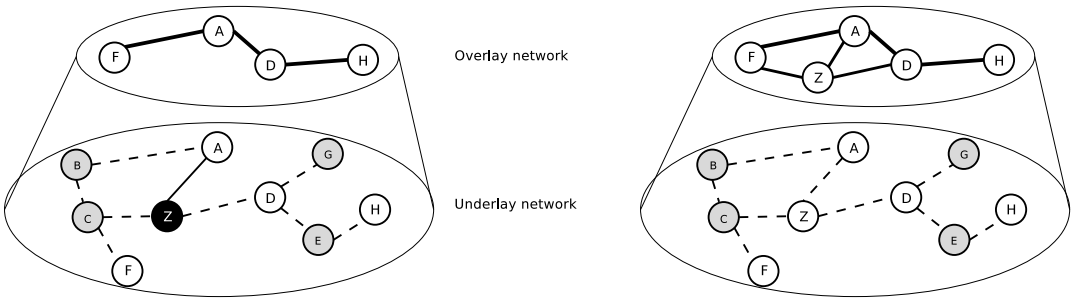
Figure 2.6: The P2P–MANET Framework

Chapter 3

Bootstrapping

Before a node can participate in a P2P overlay, it must first "bootstrap" itself into the overlay. Bootstrapping addresses the problem of joining a P2P overlay. The node must first find other nodes that are already part of the overlay and attempt to create an overlay connection with some of them. This problem has largely been ignored in the research literature, even for wired P2P networks. Without a means of bootstrapping, it is impossible to participate in a P2P overlay. Many P2P networks are designed to be fully decentralized with the exception of bootstrapping. Existing bootstrapping techniques usually require a centralized service, which is problematic in MANETs.

Figure 3.1a shows how an overlay network might exist over top of a MANET. In the figure, the white nodes represent overlay members, and the grey nodes are MANET nodes that are not participating in the overlay. There are 4 peers currently participating in the overlay network, F, A, D, and H. None of the four are directly connected in the underlay network, but through a series of logical connections they are able to communicate with one another and have the topology shown. The black



(a) Initial overlay network (b) After bootstrapping

Figure 3.1: Bootstrapping in a P2P–MANET

node, Z, is in the bootstrap phase, meaning it is attempting to become part of the overlay, but has not yet succeeded. Figure 3.1b shows how the overlay might change once Z has successfully joined the overlay. The underlay network is not affected, but the overlay network now has an additional peer and several more connections. How the connections at the overlay level should be made is the subject of the next chapter, on overlay topology control. In this chapter, we present a distributed bootstrapping mechanism that makes use of multicasting.

The remainder of this chapter is organized as follows. Section 3.1 outlines the proposed bootstrapping mechanism and gives the objectives it aims to achieve. Section 3.2 details the bootstrapping mechanism. Section 3.3 presents a performance evaluation of the proposed bootstrapping mechanism. Section 3.4 provides a summary of the chapter.

3.1 Scheme Outline and Objectives

In this chapter, we propose a bootstrapping scheme for P2P–MANETs. The scheme is designed to meet the following objectives:

- Obtain many responses per join query
- High bootstrap success rate
- Fast responses
- Low message overhead

When a node wishes to join an overlay, it will send a request to a group of nodes to determine if any are already a member of the overlay, or if they know the address of a node that is. Obtaining as many responses per join query as possible is important because more responses to the initial request provides the requesting node with more nodes to choose to connect to. If only a single node responds positively, the requestor has no choice but to connect with that node, regardless of how beneficial that is for the topology. Awareness of more overlay nodes provides the topology control algorithm with a greater selection of nodes to choose from and therefore, a better topology.

A high bootstrap success rate measures how often the requesting node is able to successfully join the overlay. If not even a single peer responds to the bootstrap query, then it is impossible for the node to join the overlay. On the other hand, a high response rate will increase the chances of connecting to at least one peer.

Fast responses to the bootstrap query allow nodes to join the overlay more quickly. This is beneficial to the user since they will be able to use the P2P overlay to acquire services from it that much faster. Low message overhead is important because reducing the amount of traffic required to join the overlay lowers the network traffic in general, and also reduces energy consumption. There is a tradeoff between the amount of message overhead and the speed of the response. In general, the more traffic that is generated, the faster the response. This traffic may be generated immediately after

the query, as with flooding, or by somehow organizing the peers beforehand, as with a multicast group. If a node sends few bootstrap queries out, it will create little traffic, but will also get few responses, if any.

Most existing P2P networks assume the availability of a list of "well–known" addresses which are usually online, or that a cache of existing addresses can be accessed from a centralized source such as a web server. MANETs are fully decentralized and as such, one cannot rely on any centralized services or a pre–existing list of well-known addresses. Another technique is Random Address Probing (RAP), in which a node randomly selects an address and sends a join request to it in the hope that it is connected to the P2P overlay.

The bootstrapping scheme proposed in this chapter works as follows. All nodes in the MANET must first join a multicast group which is used for both sending join requests and responding to them. When a node wishes to join a P2P overlay, it multicasts a MANET–wide join request. When a join request is received by an existing overlay member who is willing to accept connections to more peers, it will multicast a MANET–wide response. All MANET nodes, whether or not they belong to an overlay, will cache this response in the event that they wish to join the overlay in the future. After a sufficient time, the original requesting node will have one or more responses to select from. It can then connect to a set of neighbouring peers. The process of selecting which specific peers to connect to is the function of the topology control algorithm.

3.2 Decentralized Bootstrapping Scheme for P2P–MANETs

In a given MANET, there may be several simultaneous, extant P2P overlays. For example, there may be a file sharing overlay, a VoIP overlay, and a gaming overlay, as shown in Figure 3.2. The three overlays all make use of the same underlying mobile ad hoc network. Some nodes may be part of more than one overlay at the same time, while other nodes may not be a member of any overlay at all. The proposed scheme works regardless of how many overlay networks exist within the MANET. The requesting node may choose to join a specific overlay, or it may choose to query all overlays to determine which ones are available.

We consider a MANET in which there is at least one overlay and there exists a MANET node that wishes to join an overlay. This node may already be a member of one or more other overlays. The node may know which overlay it wishes to join and only needs the addresses of some overlay members, or it may send a general request, in which peers from all the different types of overlays respond. The request and response messages are multicasted but the bootstrapping scheme does not require a specific multicast protocol.

The information received in response messages are cached for potential later use by nodes. All nodes which receive a response will cache it, even if they are not planning to soon join a P2P overlay. Once a requesting node has received responses, it is then able to determine which nodes to connect to. This phase of the join process is part of the topology control algorithm and is discussed in Chapter 4. In this chapter, a join request is an attempt to join a P2P overlay, while a connection request is a message

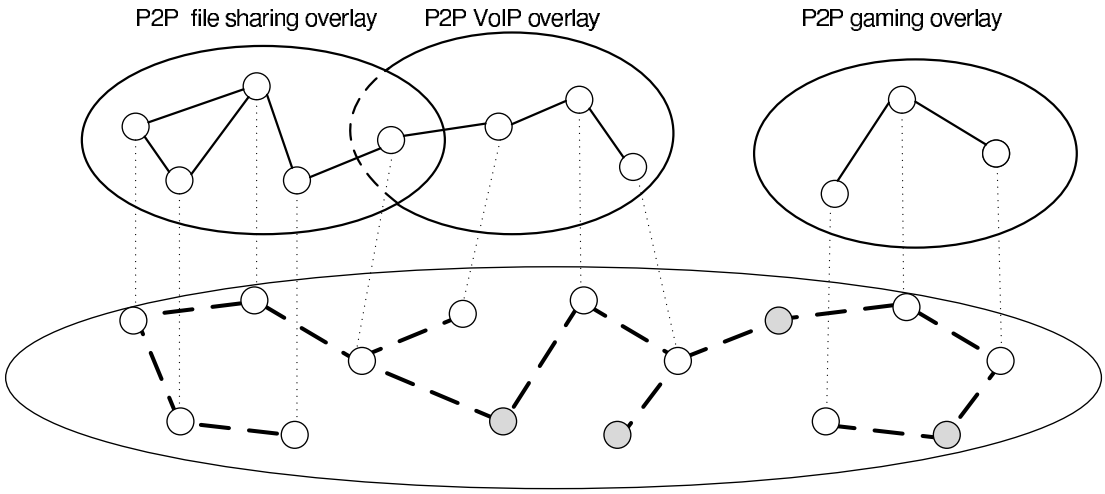


Figure 3.2: Multiple overlays on a MANET

to a specific peer to become its neighbour.

In this section, we present the details of a bootstrapping scheme that is, to the best of our knowledge, the first to address the problem of bootstrapping P2P overlays in MANETs. The scheme is completely decentralized in nature. It uses the idea of multicasting to all nodes, something that is impractical at the network layer on the Internet, but is possible in MANETs.

Nodes discover overlays by finding peers that are already members of that overlay. Suppose node c wishes to join an overlay. c first examines its local cache for previously–known peers. If none are found, or if the information is stale, c multicasts a MANET–wide join request. When a join request is received, an overlay member s_i may choose to respond to it, or to ignore it. If s_i is willing to accept a new neighbour, it will multicast a network–wide response. All MANET nodes, whether or not they belong to that overlay, cache received responses in case they wish to join an overlay in the future. After a timeout period, c will have received r responses, $S = \{s_0, s_1, \ldots, s_{r-1}\}$. c then selects which $s_i \in S$ to send connection requests to.

This selection of peers is the responsibility of the topology control algorithm. In general, however, it is desirable to choose a nearby overlay member because it would typically result in greater responsiveness and also more closely match the underlying topology of the network. Once the node has successfully connected to at least one $s_i \in S$, it is an overlay member, and has successfully bootstrapped itself into the P2P overlay.

All nodes in the MANET are members of a multicast group, used for sending join requests and responding to them. Any multicast routing protocol may be used, such as the edge–node based algorithm presented in [43], Multicast Ad Hoc On–Demand Distance Vector (MAODV) [72], or On-Demand Multicast Routing Protocol (ODMRP) [57]. For unicast messages, any unicast routing protocol may be used. The actions of nodes in the MANET for the bootstrapping algorithm are given in Algorithms 1 and 2.

Algorithm 1 shows that when a node in the MANET wishes to join a P2P overlay, it first examines its local cache. First, the cache is sorted by the time–to–live (TTL) value and stale entries are removed. If any entries remain, the cache is sorted according to the requirements of the topology control algorithm. Without loss of generality, we assume that the topology control algorithm evaluates some utility function and obtains a value. The valid cache entries are then sorted based on this value, and peers are contacted in order of utility. Precisely how the node proceeds from this point depends on the topology control algorithm. It may require that all peers be contacted or it may require a smaller number of peers be contacted.

If there are no valid cache entries, or an insufficient number of them, the node multicasts a join request (*mc–join–request* message) to the multicast group. It is possible

Algorithm 1 Joining node actions

```
1: function select-peers:
2: sort cache by TTL
3: remove expired entries
4: if cache not empty then
5:     sort cache by utility value
6:     while more peers do
7:         p ← front of queue
8:         send connection request to p
9:     end while
10:    if need more neighbours then
11:        multicast mc-join-request message
12:        timer ← T
13:    end if
14: else
15:    multicast mc-join-request message
16:    timer ← T
17: end if

18: function received mc-join-response message:
19: add reply to cache
```

Algorithm 2 Actions of other nodes

```
1: function received mc-join-request message:
2: if no available neighbour slots or wrong overlay then
3:     ignore mc-join-request message
4: else
5:     multicast mc-join-response message
6: end if

7: function received mc-join-response message:
8: add reply to cache
```

for multiple overlays to exist in the MANET, reflecting different P2P applications. If the node knows which overlay it wishes to join, this can be indicated in the initial request. Otherwise, the join request is considered generic and peers belonging to all overlays in the MANET may respond. Finally, a timer is initialized with the default timeout value, T. Once this timer reaches zero, the node proceeds as in lines 2 – 8 of Algorithm 1.

The requesting node c will occasionally receive *mc-join-response* messages. The information from these messages are added to the cache, and no further actions are taken at that time.

Algorithm 2 shows that all peers willing to accept a connection from the requesting node multicast a response (*mc–join–response* message) back to the group. This response includes any information required by the topology control algorithm, such as the peer's address, remaining energy, and hop distance. If a peer is not willing to accept any new connections, it simply ignores the request.

All devices that receive the response cache the information, even if they are already part of a P2P overlay. Existing overlay members may decide, in accordance with the topology control algorithm, that the peer response they have just received represents a better neighbour and attempt to create a new connection and possibly drop an existing one. In this case, the peer may send a unicast connection request to that peer to try and gain a new neighbour. The response reflects new network information due to node mobility or because of other peers changing their neighbours.

Cached information is maintained in a soft state, meaning that it is time limited. When a multicast response is received, it is placed in the cache with a fixed time–to–live value. This value can then be decremented as time passes, and removed from

the cache when it reaches zero. Alternatively, stale entries may be cleared only when accessing the cache, as shown in Algorithm 1. This technique is simpler and more efficient since stale cache entries are removed only when needed.

It is important to note that even though it is expected that most peers will remain in the overlay for a prolonged period, some information, such as their remaining energy is not long–lived. Therefore, a node may decide to multicast a join request, even though the cached entry is still valid. When a node caches a response message, only minimal information provided in the peer response is stored and so a large cache is not needed. If the cache is full, the new entry will replace the one with the shortest TTL value.

There are only two types of messages in the proposed bootstrap algorithm. The first is the overlay join request *mc-join-request*, which is a multicast message sent to indicate that the node wishes to join an overlay, either a specific one, or a general request. The second message type, *mc-join-response* is the response message, multicast by peers that are willing to accept new neighbours. The message sent to create a connection with a neighbour lies at the border of the bootstrap and the topology control algorithms.

The number of messages required to support this algorithm depends partly on the multicast routing algorithm used. Many routing algorithms are proactive and send messages to maintain the multicast tree. In terms of the bootstrap algorithm itself, when a node wishes to join, it multicasts its message to the $n-1$ other MANET nodes. In the worst case, $n-1$ peers multicast a response back to the rest of the network. This means a worst case message complexity of $O(n^2)$ since each node would send a message to every other node. Due to the use of caching and the fact that many

peers will not respond, the average message complexity is expected to be much lower.

3.3 Performance Evaluation

We now evaluate the performance of the proposed P2P–MANET bootstrap scheme using the network simulator *ns–2* 2.33 [68]. We first describe the simulation model and the performance metrics that are used to evaluate the scheme. We then provide a discussion of the simulation results.

3.3.1 Simulation Model

In our simulations, we use 100 MANET nodes, with the number of nodes participating in a P2P overlay varying from 50 to 100 in increments of 10. For simplicity, we assume only one P2P overlay exists in the MANET. The network area is 1500 m $\times$ 1500 m, the transmission rate is 54 Mbps, and the experiments run for two simulation hours each. The MAODV [72] and AODV [71] routing protocols are used for multicast and unicast routing respectively. Multicasting is only used for the proposed algorithm, not the comparison systems.

The two ray ground radio propagation model along with an omnidirectional antenna are used by all nodes. The random waypoint mobility model is used, with all nodes evenly distributed in the simulation area. Nodal velocities are distributed according to a uniform distribution, with a minimum 1 m/s and a maximum 3 m/s, and a uniformly distributed pause time with mean 60 s, thus mimicking a moderate walking pace with infrequent stops.

Table 3.1: Energy consumption constants used in simulations

m_{send}	1.89	$\mu W \cdot sec/byte$
b_{send}	246	$\mu W \cdot sec$
m_{recv}	0.494	$\mu W \cdot sec/byte$
b_{recv}	56.1	$\mu W \cdot sec$
$b_{sendctl}$	120	$\mu W \cdot sec$
$b_{recvctl}$	29.0	$\mu W \cdot sec$

The energy consumption model used in the simulations is the linear model proposed by Feeney [32]. Each MAC layer operation takes a certain amount of power as defined by $cost = m \times size + b$ where m is the incremental cost of the operation, b is the fixed cost, and $size$ is the amount of data sent or received. The constants are obtained by physical measurements for a Lucent IEEE 802.11 WaveLAN PC Card from [32] and are summarized in Table 3.1.

Initially the network is in a steady state, meaning all peers save one are joined initially. For example, with 50 overlay nodes, the experiments start with 49 peers joined. In each simulation experiment, one randomly selected node that is not a member of the overlay attempts to join it, and another randomly selected peer that is a member, leaves it. This process repeats every interval, which is set at 30 s. The timeout value, T, is also set to 30 s.

To the best of our knowledge, we are the first to propose a bootstrapping mechanism for P2P–MANETs. Therefore no directly comparable alternative schemes exist, and we therefore compare our scheme with two other well–known schemes, both of which are possible to use in a MANET. The first is a flooding algorithm. In this scheme, when a node wishes to join the P2P overlay, it floods the join request to every node in the MANET.

The second comparison scheme is Random Address Probing with address knowledge. When conducting experiments with the normal (i.e. "blind") Random Address Probing scheme, nodes were virtually never able to connect to the overlay at all due to the low probability of randomly selecting a valid address from a large address space and the comparatively few nodes that are part of the overlay. On the Internet, where within a subnet there may be over one hundred overlay nodes, the chances of a successful probe are much more likely. In a MANET, where there are far fewer overlay nodes across a larger address space, the RAP technique is much more likely to meet with failure. Therefore, in order to provide a useful comparison, the Random Address Probing technique was given additional information in the form of the addresses of all nodes existing in the MANET so that a valid random MANET node would be chosen each time. After three minutes without receiving a query response message, RAP gives up. We use three variations of RAP, which we name RAP–1, RAP–5, and RAP–10. The numbers indicate how many addresses the requestor contacts simultaneously. For example, in RAP–5, the node will send probes to 5 random addresses, then wait for a response. If there is no response in the timeout period T, another 5 addresses will be sent probes, and the process may be repeated until three minutes have elapsed.

We test the performance of our scheme both with and without caching enabled. When caching is turned on, it is not fixed at a pre–specified rate. Instead, we attempt to model a real MANET and determine the natural cache hit rate achieved during bootstrapping. To simulate a P2P file–sharing overlay, constant bit rate (CBR) traffic was being sent between members of the P2P overlay. The traffic consisted of 1000 byte packets sent 100 times per second and was started and stopped at random times,

between random peers.

In this chapter we are not concerned with the topology that is formed since selecting which peers to connect to is the subject of Chapter 4. Therefore, in all our simulations, a node will connect to the first five peers that respond. If the cache is found to have fewer than five peers, then a bootstrap request is multicasted.

3.3.2 Performance Metrics

The following performance metrics are used to evaluate the bootstrapping scheme:

bootstrap success rate The percentage of successful overlay join attempts. When at least one existing peer responds to a request, the node is able to successfully join the P2P overlay. A higher percentage indicates that a peer is more likely to locate and successfully join an overlay. This metric gives an idea of the effectiveness of the bootstrapping algorithm.

first response time The time until the first response is received. A faster response equates to greater user satisfaction since it means the user will be able to connect with other nodes and begin using the overlay more quickly.

average response time The average response time over all responses received.

average number of responses The average number of responses received for each join request. More responses gives the topology control algorithm greater leeway in determining the best topology.

message overhead The number of join request and response messages generated per join attempt. This includes multicast overhead or flooding data where

applicable. Beyond the small number of join and response messages that are required, it is desirable to have fewer messages because that means a lower response time and also reduces energy consumption. In a MANET, these are very important factors as they directly relate to user satisfaction and network longevity.

number of neighbours The average number of neighbours that peers had at the end of the experiment. This is an indication of how well connected and resilient the P2P overlay can be. It is related to the topology control algorithm, but since all experiments in this chapter use the same algorithm, it gives a measure of the quality of the responses received.

cache hit rate The hit rate of the cache. This applies only to the proposed scheme with caching enabled. Higher cache hit rates result in faster operation because there is no need to send out queries. Faster response times enhance user satisfaction.

3.3.3 Simulation Results

In this section, we show and discuss the simulations results for the distributed bootstrapping mechanism. The simulation results obtained in all experiments in this thesis have a 95% confidence level based on 10 independent runs. The confidence intervals are indicated by the error bars in the figures below.

In order for a node to successfully bootstrap itself into a P2P overlay, it must connect to at least one neighbour. Figure 3.3 shows the success rate for nodes trying to join an overlay. The proposed distributed bootstrapping algorithm performs very well, both with and without caching enabled. It achieves 100% success in most

cases. The flooding technique performs reasonably well and its success rate does not change much with the overlay size because the flooding mechanism contacts every MANET node. However, this is also a drawback because it creates excessive network traffic that sometimes prevents bootstrap responses from reaching the requesting node within the timeout period. This is why the flooding technique does not achieve 100% success. The RAP algorithms generally perform less well, particularly RAP–1. RAP–1 contacts very few nodes and as a result, receives very few responses. RAP–5 and RAP–10 do much better since they contact a larger proportion of nodes. As the number of overlay members increases, the RAP algorithms do better because the chances of contacting an overlay member rise. However, even though RAP–10 has a high success rate with 100 nodes, it is not 100% because some peers do not respond to requests since they have already have five neighbours by the time they are contacted. Also, the RAP algorithms' success rates have high variability, as seen by the large error bars, because of their inherently random nature.

Another important consideration when attempting to join an overlay is the amount of time that passes between initiating the bootstrapping algorithm and successfully connecting to the overlay. The time until the first received response is shown in Figure 3.4. The average time for all received responses is given in Figure 3.5. Lower response times are better because users prefer a faster response so that they can more quickly join the overlay and begin using its services.

The first response time for the flooding algorithm is quite low and stays the same for all overlay sizes. The flooding algorithm contacts all nodes, and so the first node to respond will generally do so quickly. As the overlay grows in size, more traffic is generated, which prevents some responses from getting through and reducing the

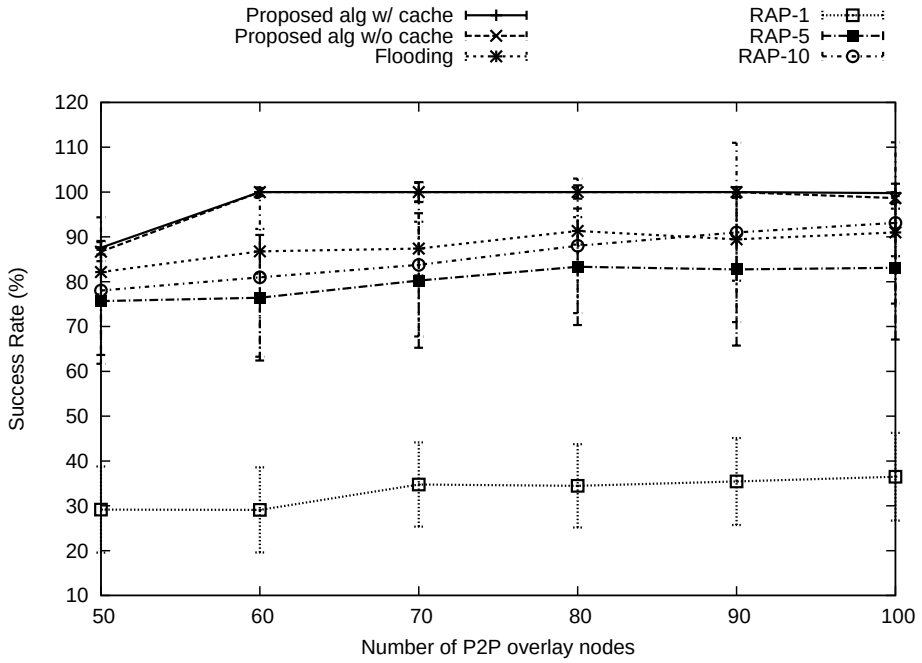


Figure 3.3: Bootstrap success rate

first response time, as occurs in all other systems. The proposed algorithm also performs very well, and as the overlay size grows the response time drops because the first responder will be closer to the requesting node. The cache–enabled version performs especially well since it may not need to send a multicast query at all. The various RAP algorithms perform the most poorly, particularly RAP–1. However, as the overlay network size increases the chances of finding a member increase, and so the first response time falls.

The average response time for all experiment types with the exception of flooding and RAP–1 increases with the increasing network size. Since all received responses are counted here, as the overlay grows in size, more responses will be received and have to travel further over the MANET, resulting in increased response times. Once again the proposed algorithm performs very well, especially with caching enabled. The

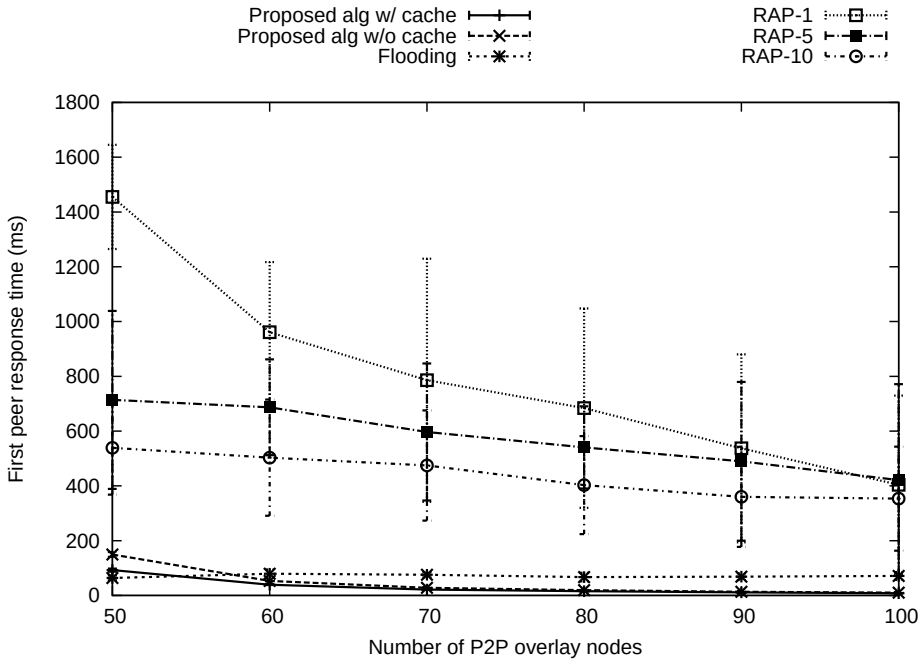


Figure 3.4: First response time

flooding algorithms' response time doesn't change because it creates so much traffic that the network is overwhelmed and more distant responses cannot get through to the requesting node. The RAP–1 time falls because only successful responses around counted. Success is more likely with larger network size. For smaller sizes, RAP–1 is mostly unsuccessful, but when it does succeed, it takes a long time to locate a valid peer.

The number of received responses to a join query is important because more responses provides the node with a greater selection of potential neighbours to choose from. If only a single response is received, then only one neighbour may be selected. However, if several responses are received, they can be evaluated by the topology control algorithm and the best neighbours chosen. Figure 3.6 shows that the proposed algorithm receives the highest number of responses per query. The uncached version

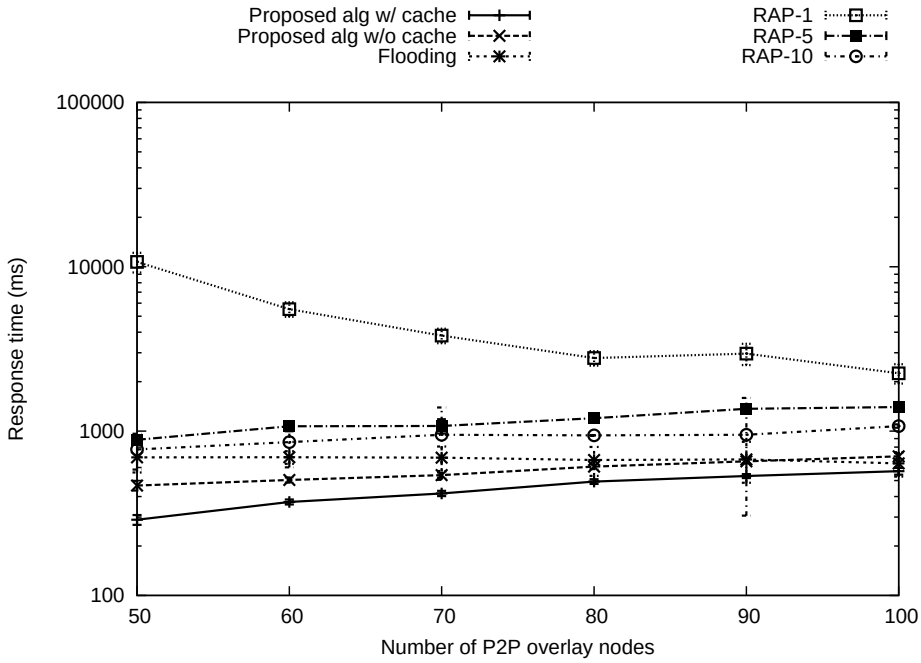


Figure 3.5: Average response time

does slightly better because the responses are more up–to–date. However, both receive many more responses than the other algorithms, particularly with larger network sizes. As explained earlier, the flooding algorithm creates so much network traffic that requests and responses are dropped, and so the number of responses remains flat with the network size. The RAP algorithms also do not change much with the network size because they will tend to contact the same number of nodes in a given time frame, regardless of the network size.

It is important to consider how many messages in total are sent by all nodes in order to join the overlay as this is a good measure of the overhead of the algorithm. Also, energy use is directly related to the number of messages exchanged, as seen in Table 3.1. Therefore, the fewer messages sent, the better. Figure 3.7 indicates how many messages are sent for each join attempt during the bootstrapping process. The

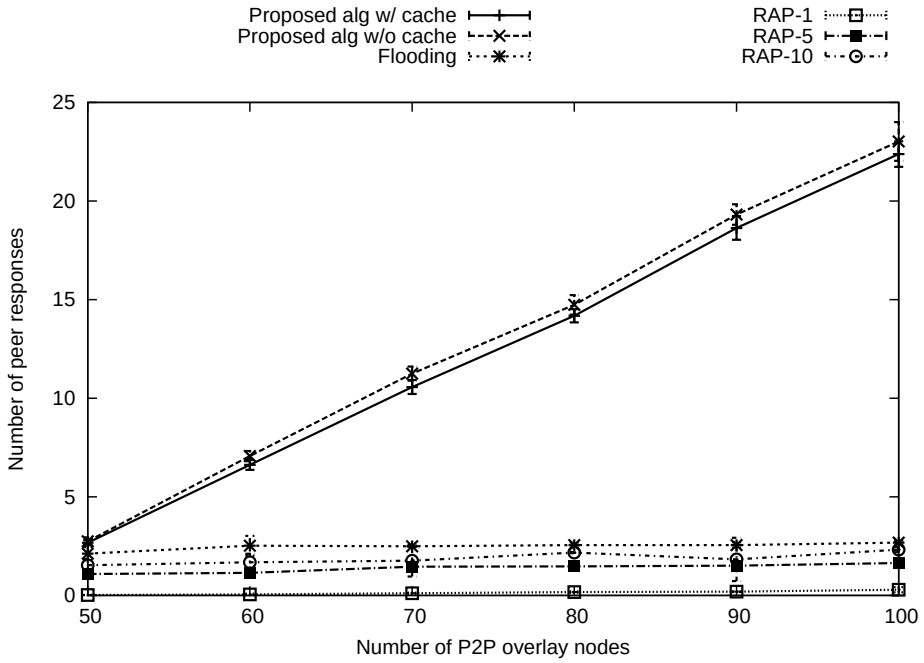


Figure 3.6: Number of responses received

overhead required by the multicast algorithm is included in the data presented for the proposed algorithm. The messages needed for the proposed algorithm increases slightly with increasing overlay size since more responses are generated. However, the bulk of the messages come from the multicast tree maintenance protocol which does not change since the MANET size is the same for all experiments. The RAP algorithms generally do best since they contact so few nodes. As the overlay increases in size and their chances of contacting a peer increase, the number of messages falls slightly. Flooding performs the worst by far, and doesn't change much with overlay size since the request must be flooded to every MANET node regardless of overlay size.

Figure 3.8 shows how many neighbours each node has on average at the end of the simulation. This number may be higher than the number of responses received

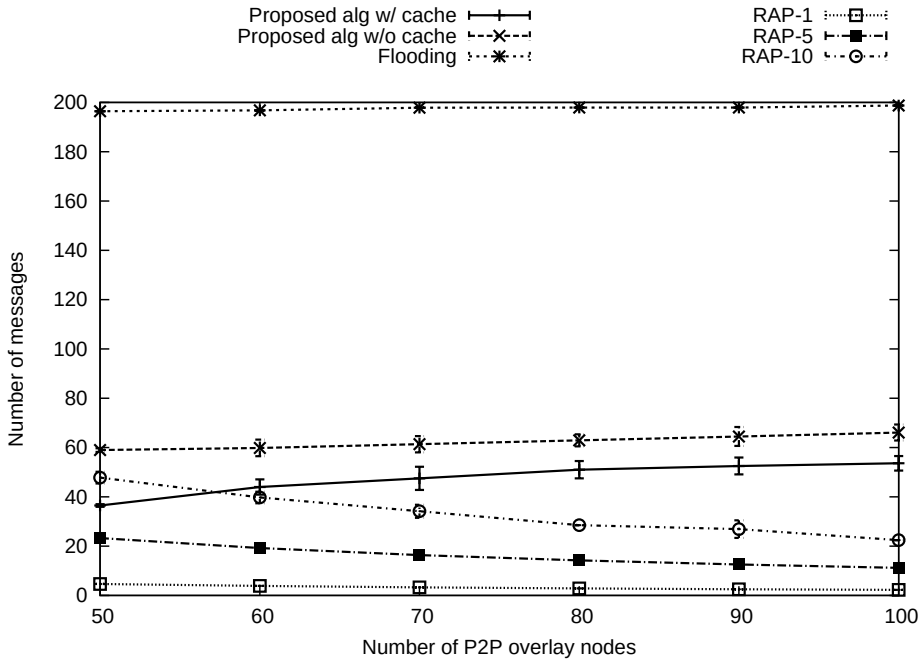


Figure 3.7: Number of messages sent

when bootstrapping because it includes both a given node's requests as well as its responses. That is, it includes the node's attempts to connect with other peers, and also other nodes' attempt to connect with it. RAP does poorly due to the low number of responses the algorithms gives, though RAP–10 performs reasonably well. The flooding algorithm performs fairly well, and the number of neighbours remains stable across overlay sizes. The proposed algorithm performs best, with the cache–enabled version doing slightly better. With a maximum of five possible neighbours, the proposed algorithm performs about as well as can be expected.

The goal of the cache is to reduce the number of message transmissions required, and as seen in the previous experiments, it is successful in doing so. Each cache miss results in a network–wide multicast query and potentially several multicast responses. In contrast, a cache hit requires only a few unicast message exchanges. Therefore, a

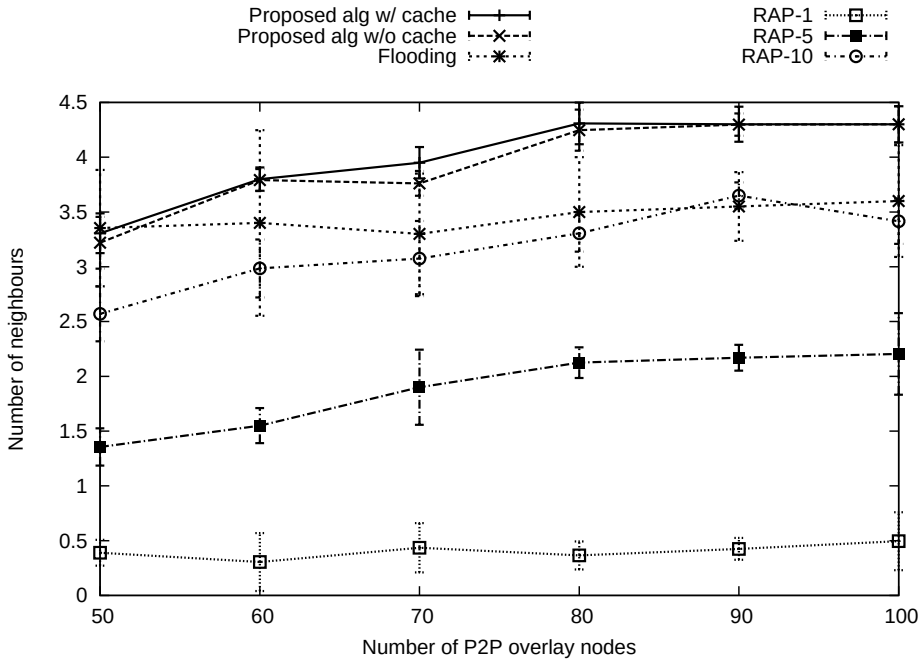


Figure 3.8: Number of neighbours

higher cache hit rate results in ever fewer multicast transmissions. In our experiments, the cache hit rate was not set at a fixed number, but instead the cache was initially empty, and was built up as would be done in a real system. Figure 3.9 shows that the hit rate increases slightly with the overlay size but flattens out slightly above 80%. This results in significant savings in message transmissions. The actual rate observed in practice depends largely on the query intervals and TTL values of the cache entries. A lower query interval means cache entries are more likely to be valid, so fewer requests are needed. A higher interval gives opposite result. Higher TTL values keep cached entries valid longer at the risk of using outdated information.

The proposed bootstrapping algorithm has been implemented as a Java application and is presented in Appendix B.

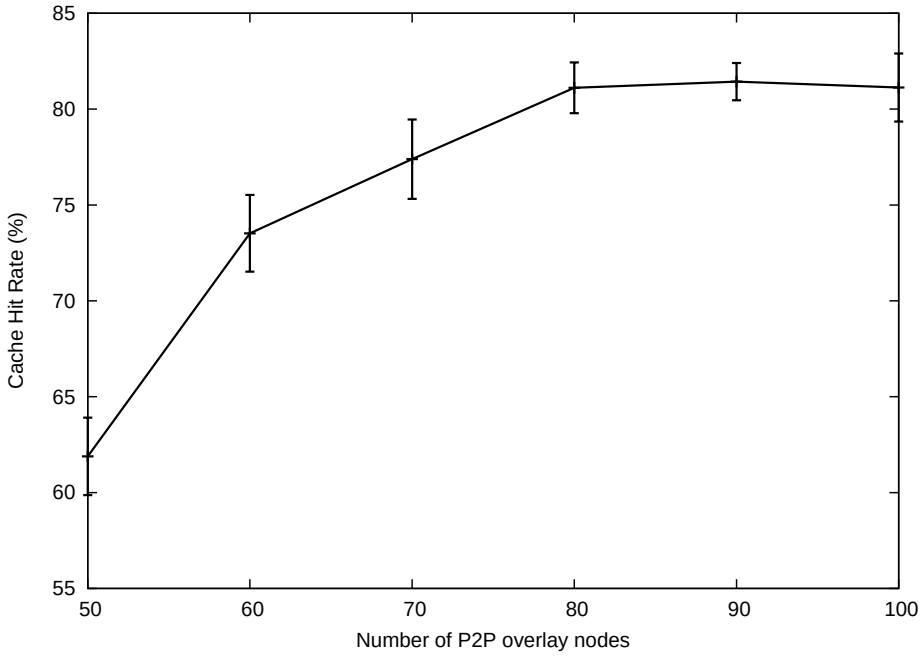


Figure 3.9: Cache hit rate

3.4 Summary

In this chapter we proposed a means of bootstrapping MANET nodes into a P2P–overlay. The scheme makes use of multicasting, which is feasible in MANETs, and supports any number of P2P overlays that may exist within the MANET. Simulation results show that the proposed scheme is able to successfully contact at least one existing overlay member more often than the comparison schemes. It also has a fast response time and generates more responses from overlay members than the comparison schemes. It has a higher overhead than the various RAP schemes, but much less than flooding. Finally, the cache hit rate appears to peak at slightly over 80%. An implementation of the algorithm shows that it works successfully in a real network, as shown in Appendix B. The next chapter presents a topology control algorithm, which is the means of connecting the peers of the overlay.

Chapter 4

Overlay Topology Control

This chapter presents an overlay topology control algorithm for P2P–MANETs. The peers of the P2P–MANET form an overlay network, which requires some topology to organize the peers. In other words, it must be determined how the various peers of the overlay are connected to one another so as to enable communication between any pair of them. The overlay must therefore be connected, meaning that any peer can send a message to any other peer, and in a MANET there may be various other goals that we wish to achieve, such as selecting longer–lived neighbours. Since all peers are distributed in a wireless network and no central authority exists, a centralized topology control scheme is not possible.

In this chapter we present a game–theoretic topology control algorithm which can be used to determine the minimum cost topology in terms of energy of neighbours and distance to other peers. We also discuss the Nash equilibrium, which is a stable topology that distributed peers may choose to form. The complexity of determining these is NP–hard, and they are thus computationally expensive to determine. We then introduce a heuristic algorithm which is computationally inexpensive, and compare

its performance to that of the minimum cost result and Nash equilibrium.

The remainder of this chapter is organized as follows. Section 4.1 presents the outline of both the optimal and heuristic topology control algorithms and discusses the objectives they aim to achieve. Section 4.2 presents the system model. Section 4.3 presents the optimal topology control algorithm in detail, while Section 4.4 presents the heuristic algorithm. Section 4.5 provides a performance evaluation of the algorithms, comparing the heuristic to the optimal and associated Nash equilibria. Finally, Section 4.6 provides a summary of the chapter.

4.1 Scheme Outline and Objectives

Once a group of MANET nodes choose to form a P2P network, the problem of organizing them in terms of a topology arises. Nodes are "closer" to some nodes than others due to the underlying physical network. Determining how peers should form logical network connections with one another is the basis of the overlay topology problem. In specific applications, it may make sense to create topologies that match the use of the overlay by its peers. However, this approach is not generic enough for all types of overlays. Therefore, in this chapter, we present a game–theoretic optimal topology control algorithm and heuristic for P2P–MANETs. The topology control algorithms are designed to meet the following objectives:

- Computationally inexpensive
- Selects longer–lived neighbours
- Reduces the distance to all other peers

In order to cope with the dynamic nature of the mobile network, it is important that the overlay topology control algorithm be computationally inexpensive so that it can be used to determine the set of links peers should have. If the algorithm requires non–polynomial time, then it may serve as a benchmark for computationally less expensive algorithms.

The benefit of selecting neighbours with greater energy remaining, those that are likely to be longer–lived, is that it provides a more stable topology. With longer–lived neighbours, peers will need to seek out new neighbours less frequently.

A topology with the set of links that gives a closer distance from a given peer to all others is preferred because it means that overlay traffic will pass over fewer links to reach its destination. This means faster delivery time and less energy consumption due to fewer intermediate MANET nodes being required to forward packets.

The game–theoretic algorithm presented in this chapter provides a minimum cost as well as a stable Nash equilibrium topology. However, computing both of these is NP–hard, and therefore it is not able to meet the computability objective. We then also present a heuristic algorithm, based on the game–theoretic model, that uses only local information and is computationally inexpensive. Both the game–theoretic and the heuristic algorithms focus on selecting neighbours that have higher energy remaining since this indicates that the neighbour will be longer–lived. Both algorithms also try to minimize the distance from a peer to others. The heuristic model, however, prioritizes the nodes which a given peer has recently communicated with, while the game–theoretic algorithm considers all peers to be equal.

The problem of finding an optimal topology for the subset of nodes in a MANET that are participating in the P2P overlay can be formulated in game–theoretic fashion

using a network creation game. The model assumes that peers are selfish and seek to minimize their own costs. This model is appropriate for MANETs because it does not require central coordination. An exhaustive search of the solution space allows us to determine the minimum cost topology, though this solution may not be a Nash equilibrium. The Nash equilibrium of the game produces a solution that is stable; one from which no peer wishes to deviate. Finding both the minimum cost solution and the Nash equilibrium of the network creation game have been shown to be NP–hard [31]. Therefore, finding the solution has non–polynomial time complexity in the number of peers, so the solution to the network creation game serves to provide a lower bound and benchmark on the cost of overlay topologies.

In the heuristic approach, we adopt a similar strategy as in the optimal solution, but do not attempt to find the global optimum. In the ideal overlay topology, each overlay link will mirror the underlay link, but in general since not all MANET nodes will participate in the P2P overlay, this will usually not be the case. In addition to simply matching the physical topology, peers have an option to tradeoff distance and try to connect to peers that have greater energy, in an attempt to have longer–lived neighbours. Another goal is simplicity of computation, so that peers can quickly determine the best neighbours. Due to node mobility and peer churn, it is expected that the topology will change quite frequently.

The game–theoretic algorithm requires *a priori* knowledge of all peers, their energy levels, and their overlay and underlay distances. With this information, it determines the entire overlay topology all at once. This means that if a single peer joins or leaves the overlay, the entire computation must be performed again. In a dynamic MANET and P2P overlay, it is expected that peers will be constantly joining and leaving. The

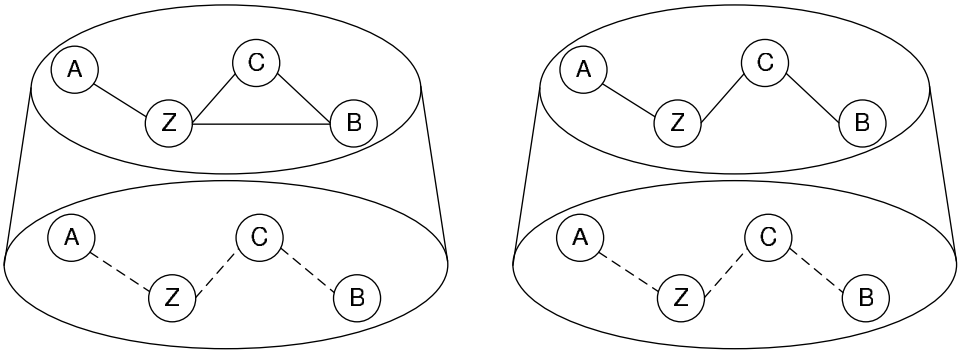
heuristic algorithm does not require complete information and builds the topology in steps, as peers join and leave.

4.2 System Model

The optimal approach follows a game–theoretic model of network creation, similar to that first proposed by Fabrikant *et al.* [31]. The game players are the overlay members, and their strategy choices create an undirected graph that represents the overlay topology. The strategy of each peer indicates which edges connect it to other peers. The union of all of these sets of edges produces the resulting undirected graph.

Clearly, the highest connectivity level possible results from a fully connected mesh network, one in which each peer is connected to every other peer. In this case, the overlay distance between peers is always one, and the underlay distance is the smallest number of physical hops between them. However, this topology results in very high overhead due to the cost of maintaining these links, and has a link complexity of $O(n^2)$.

Having fewer neighbours reduces the number of links that must be maintained and the cost that goes along with it, but increases the distance between peers. There must be a tradeoff between distance and link overhead costs. There are many definitions of distance that may be used including hop distance, delay, bandwidth, etc. In this chapter we adopt a notion of distance commonly used in P2P networks, referred to as the *stretch*. We define the stretch as the ratio between the total number of physical hops traversed and the shortest physical hop distance possible. This definition is used because in MANETs all traffic must ultimately be sent via the wireless links, in hop by hop fashion. Therefore, shorter physical hop distances are preferred since this



(a) Initial topology (b) After topology change

Figure 4.1: An example of a topology change

reduces latency and energy consumption.

The network creation game has n players, $P = \{p_0, p_1, \ldots, p_{n-1}\}$, each being a member of the P2P overlay. The strategy space of player p_i is the set $S_i = 2^{P \setminus p_i}$, representing all possible edges to all possible peers, excluding p_i itself. The strategy chosen by a peer, $s_i \in S_i$, is the set of links to be established with other peers. If $p_j \in s_i$ then peer i is establishing a link to peer j in the strategy s_i. Since the link is undirected, peer j also establishes a link to peer i. Otherwise, no link is established between i and j in the graph. The combination of the strategies $s = (s_0, s_1, \ldots, s_{n-1}) \in S_0 \times \cdots \times S_{n-1}$ form the undirected graph $G[s] = (P, \bigcup_{i=0}^{n-1}(p_i \times s_i))$.

We define $s^{-p_i,p_j} = (s_0, s_1, \ldots, s_i \setminus p_j, \ldots, s_j \setminus p_i, \ldots, s_{n-1}) \in S_0 \times \cdots \times S_{n-1}$. This is the strategy s with the link between p_i and p_j removed. We also define $s^{+p_i,p_j} = (s_0, s_1, \ldots, s_i \cup p_j, \ldots, s_j \cup p_i, \ldots, s_{n-1}) \in S_0 \times \cdots \times S_{n-1}$. This is the strategy s with a link between p_i and p_j added. We define a maximum degree, deg_{max}, associated with each peer. Peer p_i can have a maximum of deg_{max} neighbours.

Figure 4.1 shows a simple example of how the overlay topology might change in an effort to reduce the cost. In Figure 4.1a, peer Z is connected to peers A, B, and

C at the overlay level. A and C are one hop away in the underlay, and B is two hops away. In Figure 4.1b Z has dropped the connection to B and is now only connected to A and C. Z now avoids the extra cost of maintaining the link to B, but relies on C to reach peer B. Since it needs to use C to reach B in the underlay network in any case, Z has not increased the distance between itself and B, but has eliminated the overhead of maintaining that link.

4.3 A P2P–MANET Creation Game

In this section we propose a P2P–MANET creation game, based on the network creation game first proposed by Fabrikant *et al.* [31].

The cost to each peer has two components. The first component incorporates the energy level of the neighbouring peer, since it is preferable to connect to peers which have longer lifetimes. The second component is the sum of distances from the peer to all others, given that the strategy being evaluated is in place. A parameter, α, is used to capture the tradeoff between establishing a link and the change in distances to peers caused by the establishment of that link. Specifically, the cost to peer i of strategy s is

$$C_i(s) = \alpha \sum_{j \in N_i} e_j + \sum_{j=0}^{n-1} d_{G[s]}(i,j) \tag{4.1}$$

where N_i is the set of neighbours of peer i, e_j is the energy level of peer j, and $d_{G[s]}(i,j)$ is the stretch from peer i to peer j in graph $G[s]$.

The total cost is the sum of all peers' costs

$$C(G[s]) = \sum_{i=0}^{n-1} C_i(s) \tag{4.2}$$

Given Equation 4.2, it is possible to find the lowest possible total cost, which results in the optimal minimum cost topology. However, this solution may not be stable. Each peer is attempting to maximize his or her own payoff, which in this case is given by Equation 4.1. This means that if the global minimum cost is not a Nash equilibrium, peers will have an incentive to change their strategy in order to increase their payoff. The resulting Nash equilibrium, if it exists, is a stable topology because no peer will have an incentive to deviate from the set of links it produces. However, the total cost of the topology may exceed the global minimum.

A pure Nash equilibrium of this game is a strategy s such that for each player i and for all s' that differ from s only in the ith component, $c_i(s') \geq c_i(s)$. That is, no player has an incentive to unilaterally deviate from its selected strategy since this would increase its cost. This strategy is known as the best response. The difference between the minimum cost solution and the Nash equilibrium is referred to as the *price of anarchy* [31].

It has been shown in Fabrikant *et al.* that computing the best response of peer i is NP–hard [31]. This is because p_i must pick a subset of vertices from the entire graph, which is a reduction from the dominating set problem. Computing the global minimum cost is also NP–hard, for the same reason.

In an undirected overlay of n peers, there are $\frac{n(n-1)}{2}$ possible links and therefore, to find the global minimum cost, an exhaustive search must examine $2^{\frac{n^2-n}{2}}$ possible link combinations. To determine the Nash equilibrium, for each peer at each step, 2^{n-1} strategies must be examined, since for a given peer, the link to each of the other peers must be considered. These strategy spaces are exponential in size, and require exponential time complexity. Even a greedy search requires $O(n^3 \log n)$ time

[62]. Therefore, it is only practical to find the optimal solution for relatively small networks. For larger networks, the exponential time complexity strategy search must be replaced with other techniques such as random local searches.

As in Chun *et al.* [21], we make use of a random local search to examine the solution space. We adopt this strategy to find both the global minimum cost and the Nash equilibrium. It is important to note that the search may converge to topologies that are not Nash equilibria. Furthermore, there is no guarantee that a Nash equilibrium even exists.

Our random local search algorithm to find the minimum cost operates as follows and is shown in Algorithm 3. We start from a random topology and note its cost. Next, a random peer is selected, and its connection to each of the other peers is examined in turn. If the connection doesn't exist, the total cost of the entire topology is determined as if it were connected. If this results in a decrease in the total cost, then the link is established. If the connection does exist, the total cost of the entire topology is determined as if it were not connected. If this results in a decrease in the total cost, then the link is disconnected. When the change in cost results in only a small difference, ϵ, a new random peer is selected and the process is started anew. This continues until the maximum time allowed, T_{max}, has elapsed. When the algorithm terminates, the graph $G[s]$ is the overlay topology of the lowest cost graph found, and its cost is $C(G[s])$.

The random local search algorithm for determining the Nash equilibrium is similar, and is given in Algorithm 4. Again, an initial, random topology is created as a starting point. Every neighbour of every peer in the network is now examined in turn. For a given peer, p_i, the effect of disconnecting each established link on the cost to p_i, and

Algorithm 3 Random local search to find the minimum cost

```
1: G[s] ← random topology
2: C_min ← C(G[s])
3: while t < T_max do
4:    select random peer, p_i
5:    for each p_j ≠ p_i do
6:       if p_j ∈ N_i then
7:          if |C_min − C(G[s^{−p_i,p_j}])| < ε then
8:             break
9:          else if C(G[s^{−p_i,p_j}]) < C_min then
10:            N_i ← N_i \ p_j
11:            C_min ← C(G[s^{−p_i,p_j}])
12:         end if
13:      else
14:         if |C_min − C(G[s^{+p_i,p_j}])| < ε then
15:            break
16:         else if C(G[s^{+p_i,p_j}]) < C_min then
17:            N_i ← N_i ∪ p_j
18:            C_min ← C(G[s^{+p_i,p_j}])
19:         end if
20:      end if
21:   end forincrement t
22: end while
```

p_i alone, is observed. If p_i's cost decreases, the link is severed. Next, a random peer $p_k \notin N_i$ is selected. p_i then determines its cost if it were to establish a connection to p_k. If the cost decreases, the link is established.

This process is repeated for every peer in the overlay. When no peer changes any of its links, the resulting topology is considered a Nash equilibrium. Note that this topology may not be a true Nash equilibrium. The other way the algorithm may terminate is if the maximum time, T_{max} has been exceeded.

Algorithm 4 Random local search to find the Nash equilibrium

```
1: G[s] ← random topology
2: C_i^Nash ← C_i(s)
3: while t < T_max do
4:    unchanged ← true
5:    for each p_i do
6:       for each p_j ≠ p_i do
7:          if p_j ∈ N_i then
8:             if C_i(s^{-p_i,p_j}) < C_i^Nash then
9:                N_i ← N_i \ p_j
10:               C_i^Nash ← C_i(s^{-p_i,p_j})
11:               unchanged ← false
12:            end if
13:         end if
14:      end for
15:      select a random p_k ∉ N_i
16:      if C_i(s^{+p_i,p_k}) < C_i^Nash then
17:         N_i ← N_i ∪ p_k
18:         C_i^Nash ← C_i(s^{+p_i,p_k})
19:         unchanged ← false
20:      end if
21:   end for
22:   if unchanged = true then
23:      break
24:   end ifincrement t
25: end while
```

We now discuss the complexity of determining the cost of the topology of n peers.

To determine the cost for a particular peer p_i, each of the remaining $n-1$ destinations must be examined. Determining the energy of the peer requires a simple lookup. However, determining the stretch of the resulting graph requires an examination of the overlay hop distance to each of the $n-1$ possible destinations. For each destination p_j, the minimum overlay distance must be determined. This requires that each possible path from p_i to p_j be examined using the remaining $n-2$ peers as a potential first hop. Therefore, determining the total stretch of the resulting distance graph requires $O(n^3)$ time. In the case of the Nash equilibrium, each peer can calculate the distance in parallel, so the time complexity is $O(n^2)$ for each peer. The stretch must be calculated for each potential link addition or deletion in Algorithms 3 and 4.

In a MANET, where nodes are moving, the optimal solution may not remain optimal for very long. Furthermore, peers may join or leave the overlay at any time. Complete and *a priori* knowledge of peers and their status is not generally possible in a MANET. We therefore propose a heuristic algorithm which approximates the optimal solution, and which allows the topology to be built in steps. That is the topic of the next section.

4.4 A Topology Control Heuristic Algorithm

The optimal approach to the overlay topology problem suffers from a number of practical problems. The entire topology is laid down in one step, making it difficult to support peer mobility and churn. To obtain the minimum cost requires *a priori* global knowledge of all MANET nodes and the underlay hop distances between them. Determining a Nash equilibrium, i.e., a stable topology, while able to be calculated in distributed fashion, is unlikely to result in a minimum cost overlay, and may not

even be possible to obtain. Finally, the computational complexity of calculating the global minimum cost and Nash equilibrium topology requires non–polynomial time. In a P2P network, peers will constantly be joining and leaving the overlay, so this approach to topology control is impractical, though it serves as a useful benchmark. Another approach is to try and incorporate new peers as they join, and restrict the maximum degree, deg_{max}.

The heuristic algorithm presented utilizes a two–phase mechanism for each peer and uses a slightly modified cost function. In the first phase, a MANET node chooses the initial neighbours it wishes to connect to. From the node's point of view, this is the culmination of the bootstrapping process. Once successfully joined to the network, it actively maintains its links, changing them as needed to decrease its costs. This second phase performs the topology maintenance.

The cost function used in the heuristic is similar to Equation 4.1, except that only the distance to a subset of P is determined

$$C_i^{heur}(s) = \alpha \sum_{j \in N_i} e_j + \sum_{j \in H_i} d_{G[s]}(i,j) \tag{4.3}$$

where N_i is the set of neighbours of peer i, e_j is the energy level of peer j, $d_{G[s]}(i,j)$ is the stretch from peer i to peer j in graph $G[s]$, and H_i is a subset of P referred to as a *hot list*. The hot list consists of the deg_{max} peers that p_i sends and receives the most messages to and from, and is explained further later in this section.

When a node n_i wishes to join the overlay, it determines which peers it would prefer to connect to. The information needed to make this determination should be collected during the bootstrap phase, as discussed in Chapter 3. The set of best peers for a node to join is determined by the Cobb-Douglas utility function [82]

$$P = h^{\beta} \times e^{1-\beta} \tag{4.4}$$

where h is the hop distance of the MANET node in the substrate network, and e is the energy of the potential neighbour. Both h and e are normalized, and β is a parameter indicating the preference of a nearer peer to a longer–lived one. When $\beta = 0$, this indicates that the node wishes to select the peer with the most remaining energy regardless of its distance. This may result in an overlay topology which does not closely reflect the underlying network topology, however the connection to the peer is likely to remain for a longer period of time. On the other hand, when $\beta = 1$, this indicates that the node wants a neighbour that is closer, no matter what its energy level is. The benefit of matching the overlay topology to the underlying network is that fewer hops are required when overlay neighbours communicate, which results in lower delay and network–wide reduced energy usage. n_i cannot use Equation 4.3 because it does not know how the topology is connected and therefore cannot determine a value for the stretch.

If the node is the first in the overlay, it is bootstrapped as the sole peer. Nodes that join later proceed as follows.

When a node n_i wishes to join an overlay network, it must connect to at least one overlay member, and preferably several more. The utility function (Equation 4.4) is applied to all known peers in the overlay, and the list is sorted. n_i then contacts the deg_{max} top peers and requests a connection with them, one by one. When a peer p_j is contacted by n_i, it evaluates Equation 4.3 and will agree to connect to n_i only when the resulting cost is less than or equal to its current cost. Peers are associated with a maximum degree, deg_{max}, so if p_j already has deg_{max} neighbours, it will also reject

n_i's request.

The very first peer p_j that n_i contacts will always agree to the connection request, provided it doesn't result in exceeding deg_{max}. This is necessary because n_i will not be on any peer's hot list as it is joining, always resulting in a greater cost to p_j. Therefore, to successfully bootstrap n_i, one peer must agree to allow n_i to connect to it. n_i makes the connection requests one by one because this provides an accurate picture of its connectivity to the potential neighbours. Once it has contacted the top deg_{max} peers, n_i, now p_i, has successfully bootstrapped itself into the overlay if it has at least one neighbour. At this point, the second phase of the heuristic algorithm, topology maintenance, begins for p_i.

For the lifetime of the overlay, peers verify a link's status via a mechanism similar to Gnutella's "ping–pong" technique. During this process, p_i sends information to all $p_j \in N_i$ to inform them of the remaining energy level and stretch to all other peers p_i knows of. p_i may also send these messages to its neighbours when it learns of new information, such as a change in link status or a new peer that has joined the overlay.

Given the new information received by p_i from its neighbours, p_i will execute Algorithm 5 in an attempt to lower its cost. This algorithm serves to provide topology maintenance, and works as follows. For each $p_j \in N_i$, p_i examines the change to its cost if it were to drop p_j as a neighbour. If the cost decreases, the link is severed, otherwise it is maintained.

p_i also maintains a hot list, which contains information regarding how much data has been transferred between itself and every other peer in the last epoch. The epoch is defined as the time period between ping–pong messages. This list is sorted such that the peer with which p_i has sent and received the most messages, labelled as h_0,

appears at the top, and the peer with which p_i has communicated with the least, labelled as h_{n-1} appears at the bottom. The list is then truncated to size deg_{max}.

If $h_j \notin N_i$, p_i will determine its cost if h_j is added to N_i. If p_i's cost drops, then h_j is added to N_i, providing $deg_i < deg_{max}$. One possible downside to focusing on the hot list for potential neighbours is that it may result in cliques which are poorly connected to one another. Therefore, p_i will also attempt to connect to the nearest peers by underlay hop distance which are not already its neighbours. If connecting to $p_m \notin N_i$ such that p_m is within η underlay hops results in a reduced overlay cost to p_i, and does not cause deg_{max} to be exceeded, p_m is added to N_i.

Finally, to allow the chance discovery of better neighbours, p_i will select a random peer p_r that it has not yet considered in the process already described, and evaluate the cost of connecting to it. Once again, if it results in a reduced cost, p_r is added to N_i. This process allows the topology to be dynamically updated with the most recent information, providing resilience to peer churn and node mobility, and also allows p_i to focus on the peers it communicates with most.

As discussed in Section 4.3, the time required for each peer to determine the stretch is $O(n^2)$. Therefore, a faster way of determining the cost is desired. The heuristic algorithm uses Equation 4.3, which calculates the cost to the top deg_{max} peers of the hot list only, for each peer. This results in a reduced time complexity of $O(n \times deg_{max})$ to determine the stretch. Note that in the performance evaluation section of this chapter, the total cost shown is calculated over the cost of the entire overlay, not just the hot list, in order to make for a fair comparison.

The messages exchanged by the topology control heuristic include connection and disconnection messages, as well as periodic ping/pong messages to verify the link

Algorithm 5 Overlay topology maintenance

Require: $C_i^{min} \leftarrow$ current cost for p_i
  sort and truncate p_i's hot list
  for each $p_j \in N_i$ **do**
    if $C_i(s^{-p_i,p_j}) < C_i^{min}$ **then**
      $N_i \leftarrow N_i \setminus p_j$
      $C_i^{min} \leftarrow C_i(s^{-p_i,p_j})$
    end if
  end for
  for each $h_j \notin N_i$ **do**
    if $C_i(s^{+p_i,h_j}) < C_i^{min}$ and $deg_i < deg_{max}$ **then**
      $N_i \leftarrow N_i \cup h_j$
      $C_i^{min} \leftarrow C_i(s^{+p_i,h_j})$
    end if
  end for
  for each $p_m \notin N_i$ within η hops of p_i **do**
    if $C_i(s^{+p_i,p_m}) < C_i^{min}$ and $deg_i < deg_{max}$ **then**
      $N_i \leftarrow N_i \cup p_m$
      $C_i^{min} \leftarrow C_i(s^{+p_i,p_m})$
    end if
  end for
  select random $p_r \notin N_i$
  if $C_i(s^{+p_i,p_r}) < C_i^{min}$ and $deg_i < deg_{max}$ **then**
    $N_i \leftarrow N_i \cup p_r$
    $C_i^{min} \leftarrow C_i(s^{+p_i,p_r})$
  end if

status. These messages piggyback information regarding energy status and overlay distances to other peers. These distances, in conjunction with information determined by the routing protocol, are used to compute the stretch.

4.5 Performance Evaluation

We now evaluate the performance of our proposed heuristic overlay topology control algorithm, and compare its performance to the optimal P2P–MANET creation game described in Section 4.3. The P2P–MANET creation game was evaluated in a solver written in the C++ programming language. The heuristic algorithm was implemented in the network simulator *ns–2* 2.33 [68]. The resulting underlay topologies were then fed into the P2P–MANET creation game solver to determine the optimal overlay topology for the simulated networks, ensuring consistency of the results. We now discuss the simulation model and performance metrics, followed by a discussion and comparison of the results.

4.5.1 Simulation Model

For consistency, the simulation model used in this chapter is the same as the one used in Chapter 3. The input topology to the P2P–MANET creation game is the underlay topology at the end of the simulation along with the resulting energy values. The bootstrapping mechanism discussed in Chapter 3 is used to initially discover peers on the network, though any bootstrapping scheme can be used. Peers join the overlay at 30 second intervals and use the heuristic algorithm to select their neighbours from the responses received.

Constant bit rate (CBR) traffic of 1000 byte packets sent 100 times per second was sent between members of the P2P overlay. The traffic started and stopped at random times, between random peers. This created overlay traffic, which was used to determine peers' energy levels, as well as the hot list information.

The performance of the heuristic algorithm is compared to the results received from the P2P–MANET creation game. The results of the Nash equilibrium are also presented, though a Nash equilibrium could not be found in all experiments. The underlay networks from the simulations were used as inputs for the optimal topology solver. However, as discussed in Section 4.3, obtaining the actual minimum cost for an overlay by performing an exhaustive search of the solution space is infeasible for the network sizes examined here, 50–100 overlay nodes. We therefore perform a random local search of the solution space using Algorithms 3 and 4. An initial random point of the solution space is examined, and we proceed to walk to progressively better results in the area as discussed in Section 4.3. When the difference between successive results is less than 1%, we jump to a new random point of the solution space and begin walking again. The minimum cost and Nash equilibria are not constrained by a maximum degree in our experiments, i.e., $deg_{max} = n$.

The α parameter is set to the values 1, 10, 100, and 1000 to determine its effects. The P2P–MANET creation game software was run for a maximum period of 24 hours for each experiment to determine the lowest cost, and another maximum period of 24 hours to determine the Nash equilibrium. Peers' energy levels were initially set to a maximum value of 100,000 Joules, which decreased as the CBR traffic flowed. The epoch value was set to 60 seconds. Peer mobility proceeded as described in Chapter 3.

4.5.2 Performance Metrics

The following performance metrics are used to evaluate the bootstrapping scheme:

total cost The total cost of the selected overlay, $C(G[s])$ from Equation 4.1. This is the metric that is to be minimized.

relative mean error We define the relative mean error (RME) as the relative error of the mean heuristic cost with respect to the optimal value. That is, $\frac{C_{heuristic}-C_{optimal}}{C_{optimal}}$.

total stretch The total stretch for each peer in the overlay, from Equation 4.1. This metric is a component in the total overlay cost.

neighbour energy The average energy consumed for the neighbours of a peer. Lower values indicate lower energy consumption, and are thus preferred.

number of neighbours The average number of neighbours a peer has. More neighbours are preferred, up to a point. Beyond this, the complexity of maintaining neighbour links may outweight the benefit.

***K* metrics** These are a measure of the topology's resilience [70]. The K is the ratio of all connected peer–pairs in the network divided by the total number of distinct peer–pairs in the network. We refer to this value, calculated for the overlay topology selected, as $K\ metric_{normal}$. We call $K\ metric_{random-failure}$ the K when 10% of peers are randomly selected and removed from the topology. This measures the effect of a failure of a random 10% of the peers, due to effects such as peer mobility, voluntarily leaving the overlay, or running out of energy. Finally, we call $K\ metric_{popular-failure}$ the K when the 10% of the peers with

the highest degree are removed. This measures the effect of excessive traffic on the most popular peers, which may result in their running out of energy, or their leaving the overlay to prevent a large drain on their battery.

4.5.3 Simulation Results

In this section, we present the simulations results for the P2P–MANET creation game, showing both the minimum cost and Nash equilibrium, where it could be obtained. We compare these results to our heuristic algorithm, with values of 5, 10, and 20 for deg_{max}. The simulation results obtained in all experiments in this chapter have a 95% confidence level based on 10 independent runs. The confidence intervals are indicated in the figures below. Though overlay sizes of 50, 60, 70, 80, 90, and 100 nodes were tested, due to the similarity in the results, only the values for 50 and 100 overlay peers are given below.

In more than half the experiments shown, no Nash equilibria could be found within the time constraints. Even in those cases where Nash equilibria were found, no more than 6 out of the 10 test runs were successful in determining the equilibrium value. This indicates that finding the Nash equilibrium is difficult at best, and may not even exist in most cases. If no Nash equilibrium exists, then the peers will not converge to a stable topology since peers will always have an incentive to change their neighbours. This may not be a major issue for a P2P–MANET in which peers are both mobile and churning. The Nash equilibria are provided for those cases where found, but it is impossible to determine the price of anarchy for the other cases.

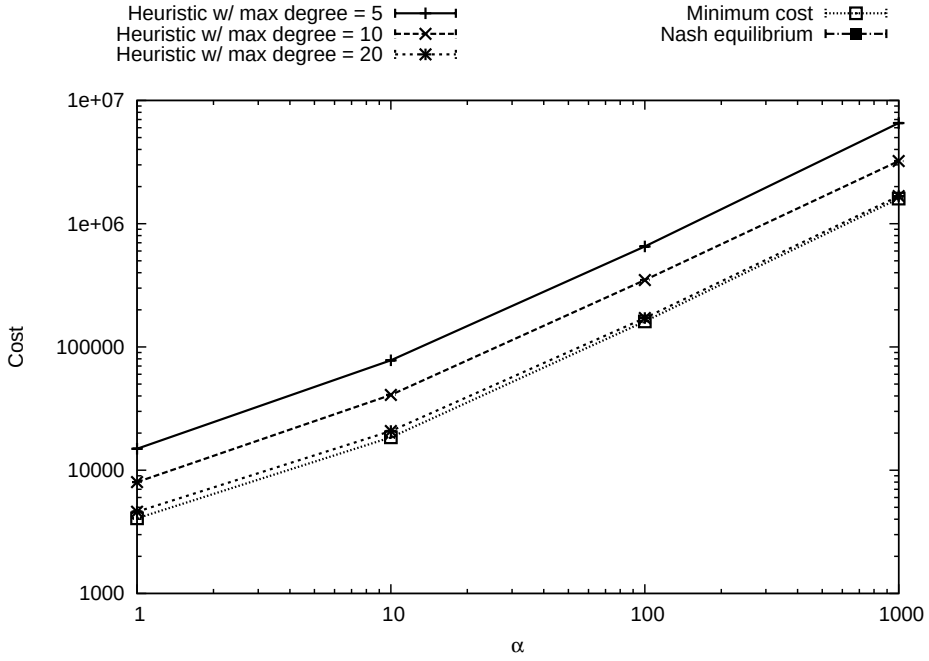
The total cost, $C(G[s])$ for various values of α and different overlay sizes are provided in Figure 4.2. The figures show that the heuristic algorithm with $deg_{max} =$

20 performs very well relative to the minimum cost P2P–MANET creation game. As the value of deg_{max} decreases, the results move further away from the optimum value, as is expected. The Nash equilibrium was not found in every case and for the overlay size of 50 peers, no Nash equilibria were found for any value of α. For those cases where a Nash equilibrium was found, it tends to be closest to the $deg_{max} = 10$ case. This indicates that the minimum cost result does not converge to a stable topology.

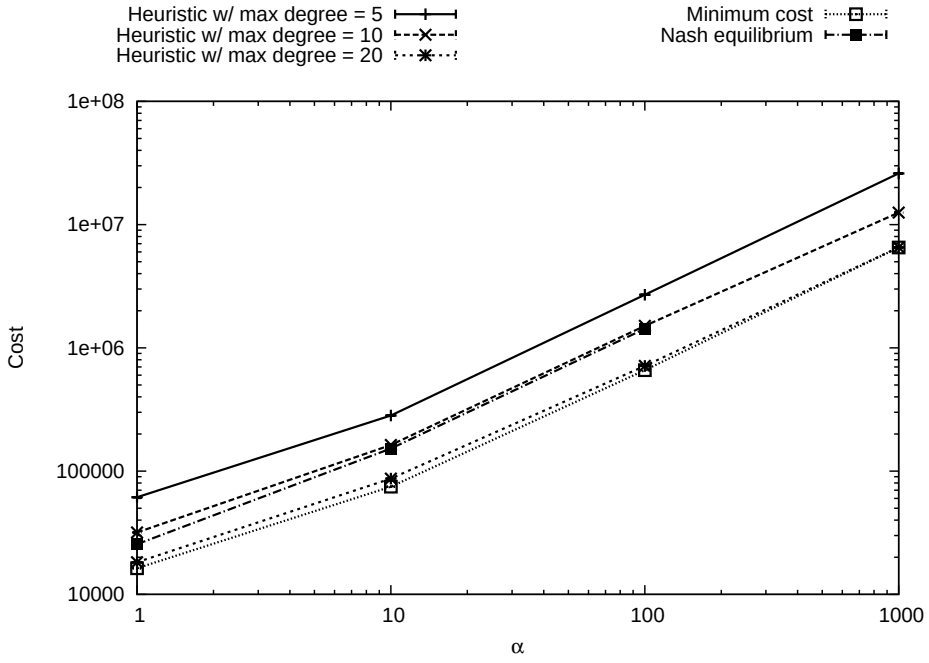
Figure 4.3 provides the relative mean error for the heuristic case as compared to the minimum cost. It can be seen that the RME tends to be fairly stable over all values of α and overlay sizes, indicating a constant difference in cost. Setting deg_{max} to higher values better approximates the minimum cost topology result.

Table 4.1 provides the price of anarchy for those cases where a Nash equilibrium could be found. The price of anarchy is defined as the ratio of the Nash equilibrium cost over the minimum cost solution. The price was found to be between 1 and 2, indicating that the stable topology may be as much as twice the cost of the minimum cost one, with no discernable trend. There is no general algorithm that finds Nash equilibria, so a random search of the solution space, as was done in this chapter, will produce unpredictable results, as seen both by the results in the table and also those that do not appear since they were not found.

Figure 4.4 shows the total stretch values obtained. This is a component of the total cost figure, but is worth examining on its own as well. As α increases, the cost of maintaining more links increases, and so the heuristic algorithm begins reducing the number of neighbours. This results in the stretch increasing because more hops are needed on average. This analysis also applies to the Nash equilibrium, though it tends to keep more neighbours than the heuristic algorithm, resulting in a stretch

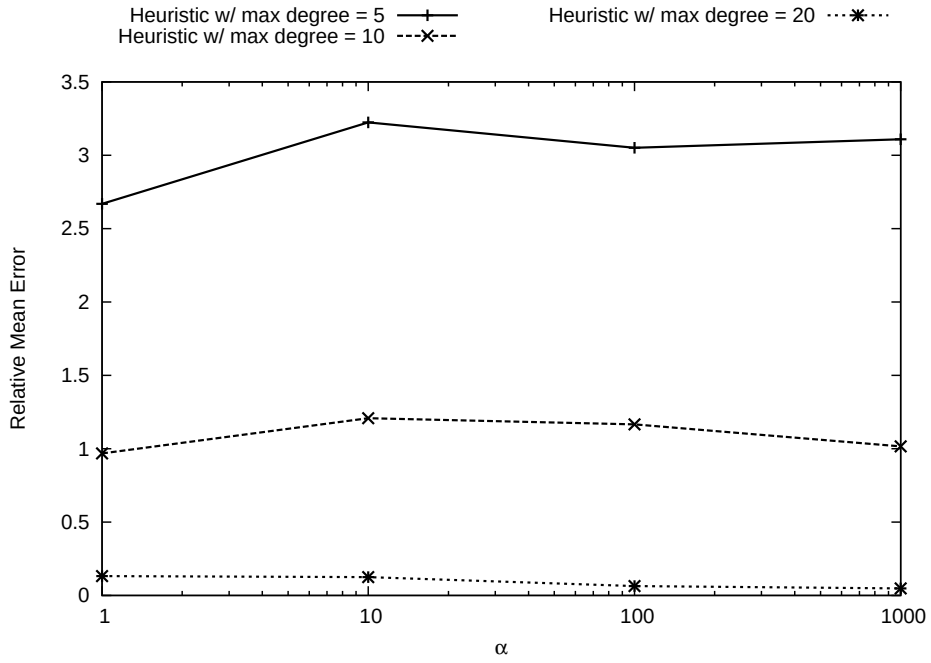


(a) 50 peers

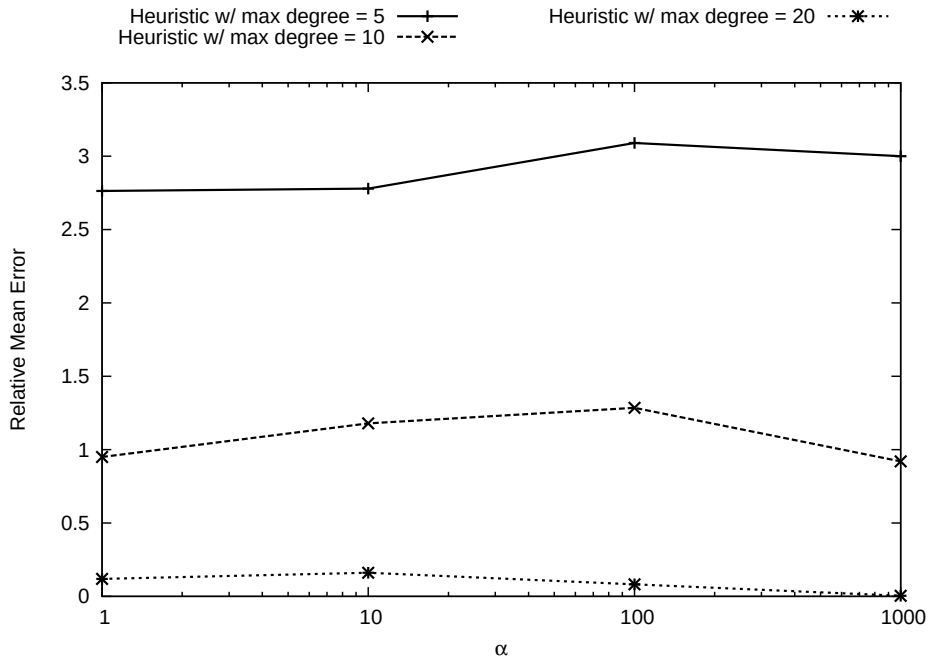


(b) 100 peers

Figure 4.2: Total cost



(a) 50 peers



(b) 100 peers

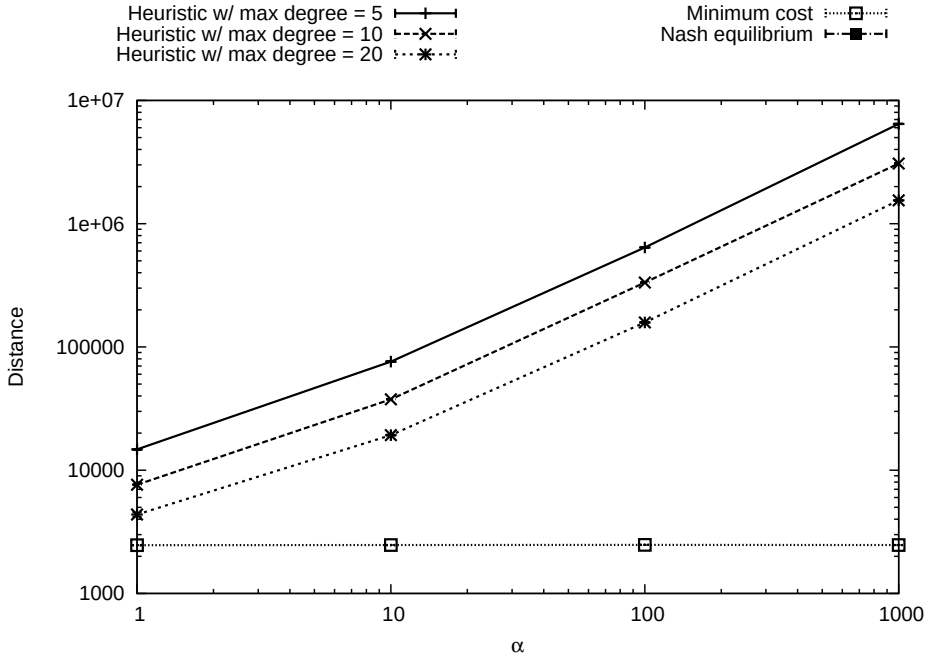
Figure 4.3: Relative mean error

Table 4.1: The price of anarchy

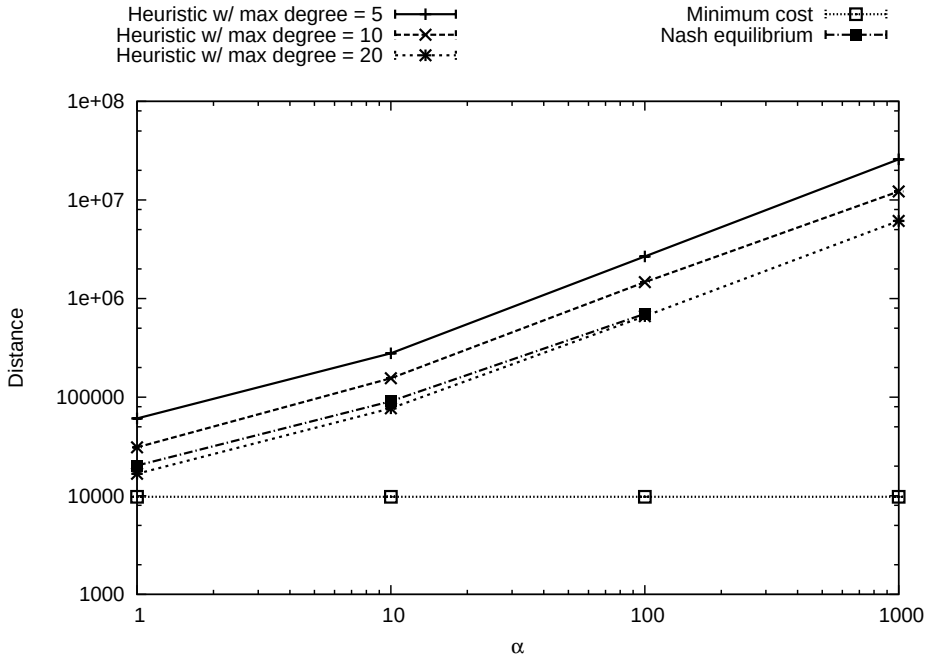
$n = 60, \alpha = 1$	1.35
$n = 60, \alpha = 100$	1.29
$n = 60, \alpha = 1000$	1.08
$n = 70, \alpha = 1$	1.88
$n = 70, \alpha = 100$	1.31
$n = 80, \alpha = 1$	1.64
$n = 80, \alpha = 100$	1.62
$n = 90, \alpha = 1$	1.47
$n = 100, \alpha = 1$	1.37
$n = 100, \alpha = 10$	1.58
$n = 100, \alpha = 100$	1.63

that is closest to $deg_{max} = 20$. Interestingly, the minimum cost topology maintains a very large number of neighbours, even though the cost increases. This results in the stretch value not changing as α increases. In fact, the stretch of the minimum cost is very close to the minimum possible stretch. This results in a very highly interconnected network, in which most peer–pairs are connected.

Figure 4.5 provides the average normalized energy consumed for each peer's neighbours. Lower values mean that the peer is connected to peers with more energy remaining. The heuristic algorithm tends to have longer–lived neighbours, mainly because the minimum cost algorithm connects to nearly all peers. As α increases, the minimum cost algorithm does not change, while the heuristic tends to connect to fewer, but longer–lived neighbours. This happens because when α increases, it becomes more costly to maintain neighbours, so the heuristic tries to maintain connections to only the most important peers, as defined by the hot list. As the overlay size increases, the energy consumed by each node increases on average due to the larger amount of network traffic. As a result, the average neighbour energy consumed rises with increasing overlay size.

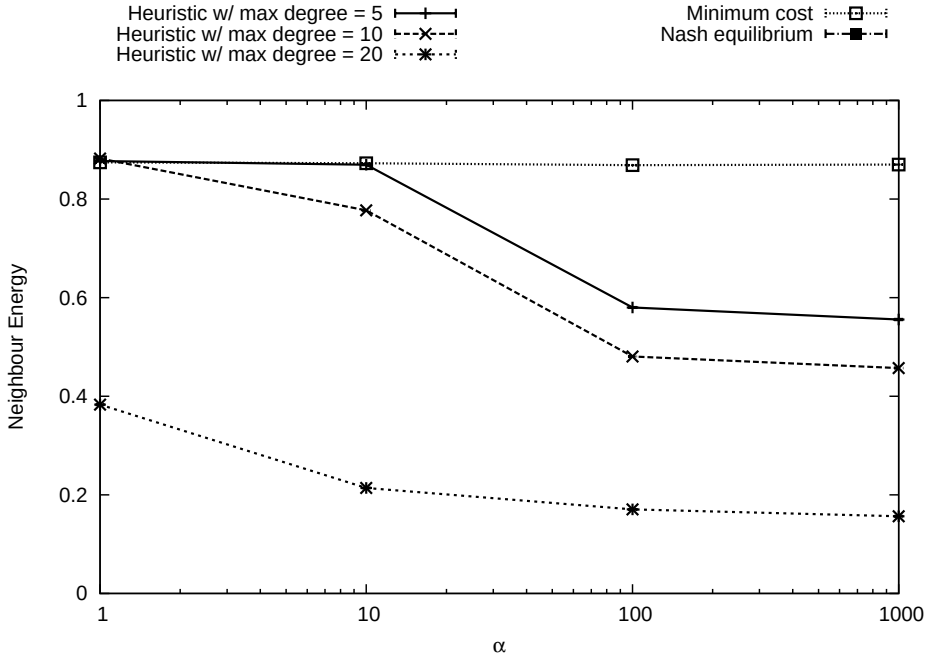


(a) 50 peers

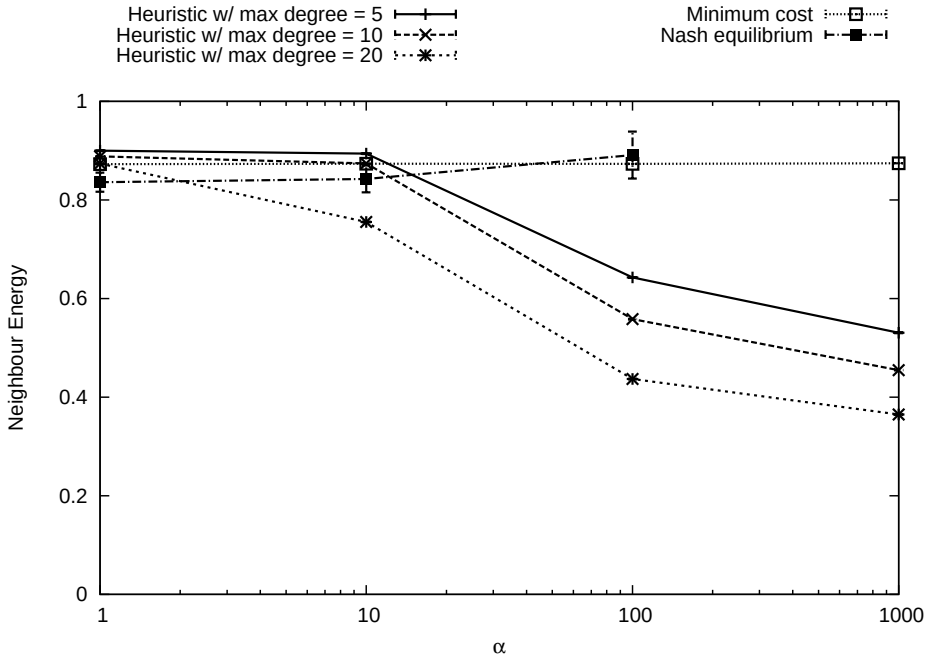


(b) 100 peers

Figure 4.4: Total stretch



(a) 50 peers



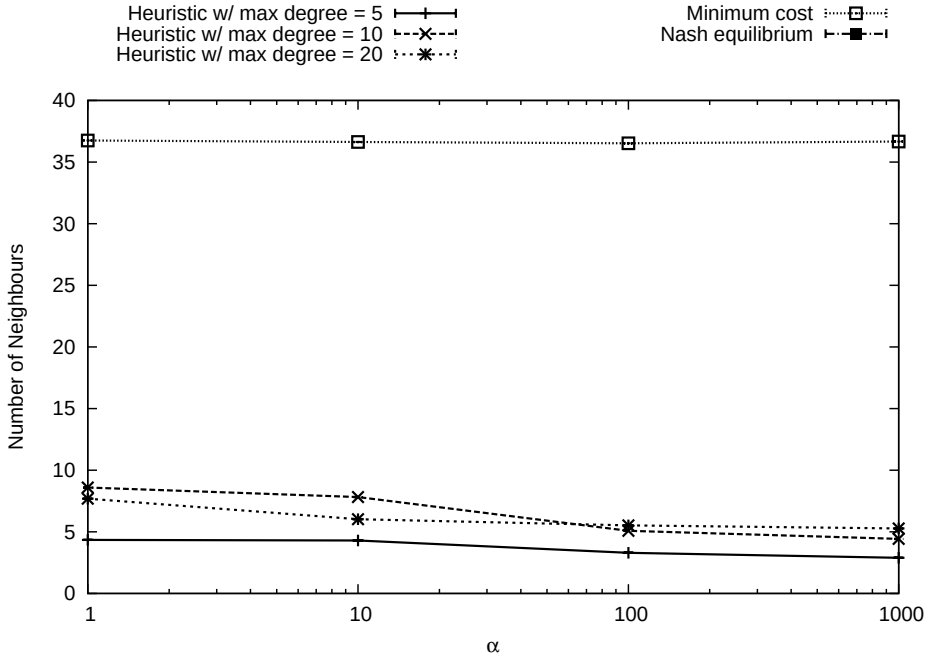
(b) 100 peers

Figure 4.5: Average neighbour energy

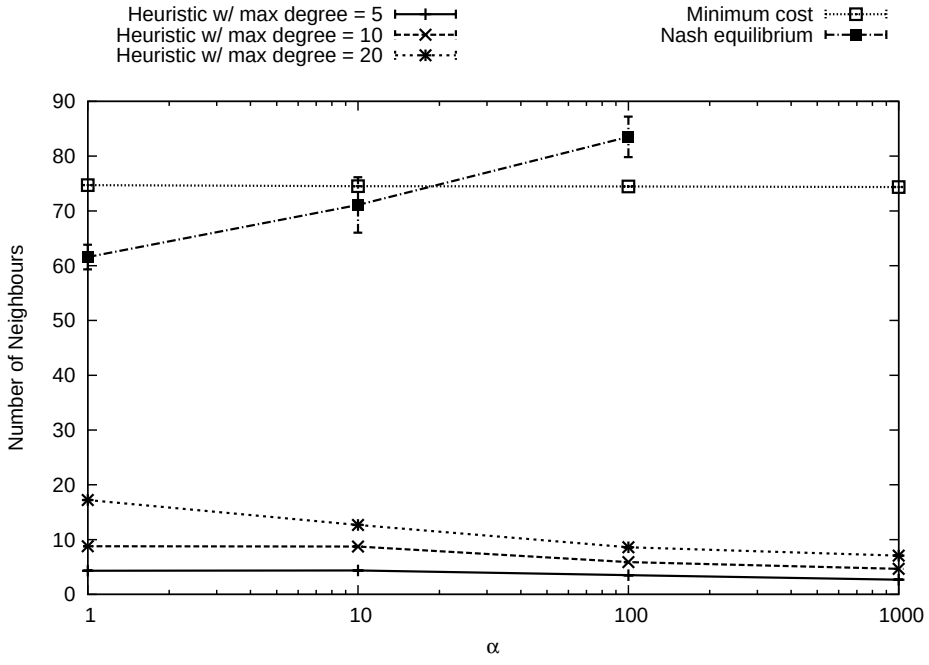
Figure 4.6 shows how many neighbours each peer has on average. As evidenced in previous figures, the minimum cost algorithm tends to maintain a very high number of neighbours, regardless of α. The Nash equilibrium figures available show that it also maintains a high number of neighbours, which actually increases as it becomes more costly to maintain neighbours. This indicates that the peers are penalized for being selfish, because it would be expected that they would maintain fewer neighbours. But only a small number of them can do so before the others are forced to compensate by increasing their degree, resulting in a higher average degree. The Nash equilibria also have larger confidence intervals due to the more volatile nature of the equilibria. The heuristic algorithm has close to deg_{max} peers when α is low, but as expected, this number falls as α increases since the cost of maintaining that many peers is too high. As the overlay size increases, the heuristic algorithm keeps more neighbours because there are more to choose from that result in a lower cost, while the minimum cost algorithm does not change.

The next three figures provide the K metric values. Figure 4.7 shows the K metric for the topologies that were determined. A value of 1 means that each peer–pair is connected, as would be the case in a fully connected mesh. We see that the minimum cost algorithm has a very high K metric because such a large share of the peers are interconnected. The heuristic algorithm, restricted by deg_{max} had much lower K values, with lower deg_{max} values resulting in lower K values, as expected.

Figure 4.8 shows the K metric assuming that a random 10% of the peers fail, possibly due to high energy consumption, peer mobility, or voluntarily leaving the overlay. Here, the minimum cost algorithm shows a reduced K metric, but as is expected, the value is still very high due to the very high level of connectedness of the



(a) 50 peers



(b) 100 peers

Figure 4.6: Average number of neighbours

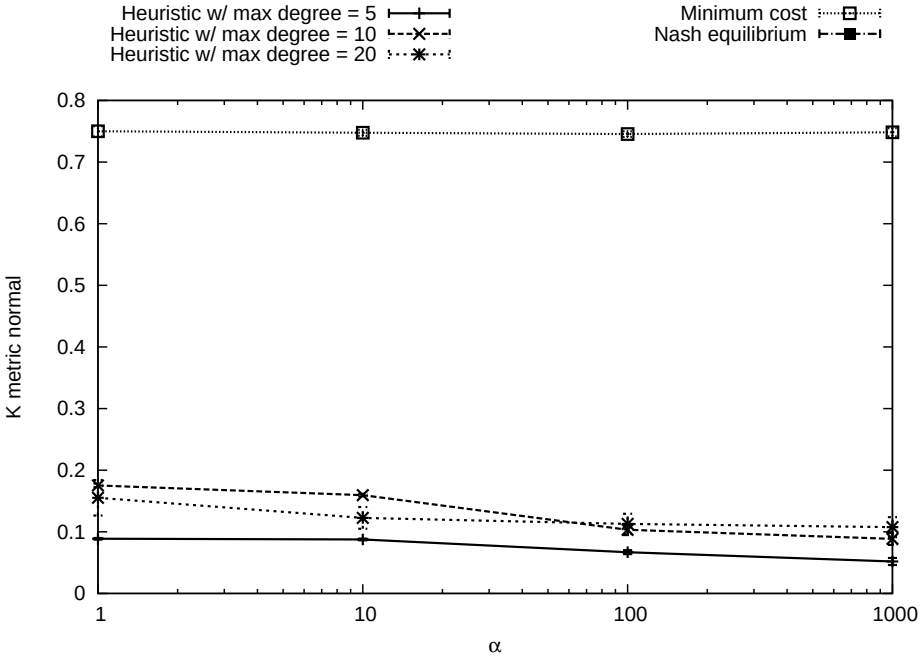
topology. The heuristic algorithm also shows a reduced K metric, as expected, with the reduction in value very slight. Because the heuristic maintains fewer connections, a given peer is less likely to be affected by a random peer failure.

Figure 4.9 shows the K metric under the scenario that the most highly connected 10% of peers are removed from the overlay. This can occur because they are required to forward the most traffic for others, resulting in excessive energy consumption. Once again, the minimum cost algorithm performs very well because since nearly all peer–pairs are connected, the effect of removing the most connected 10% is minimal. Furthermore, these values are all unchanging with α. The Nash equilibria, as expected from previous results, show an increasing K metric with increasing α since more peers become connected. This results in a K metric that is sometimes higher than the minimum cost algorithm. The heuristic algorithm has a fairly low K metric, due to the deg_{max} constraint. As popular peers leave, the K metric falls since the relatively small number of connections falls further.

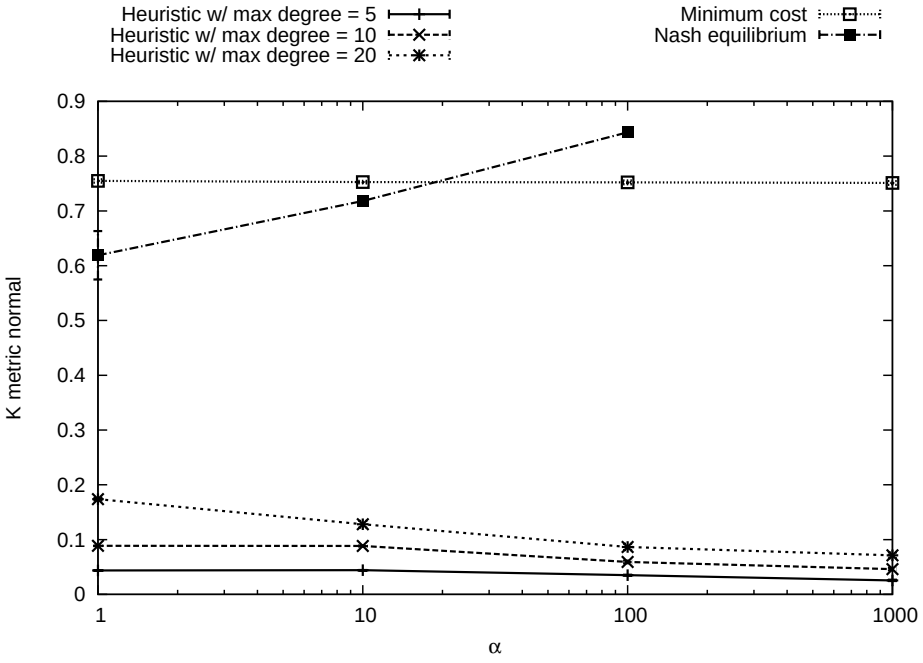
Overall, the heuristic algorithm has performed very well relative to the minimum cost, particularly with higher deg_{max}. It resulted in a similar total cost, even though it maintained far fewer neighbours. This resulted in poorer K metric figures, demonstrating weaker resilience, though this could be addressed by raising the value of deg_{max}.

4.6 Summary

This chapter provided an examination of a heuristic overlay topology control algorithm for P2P–MANETs and compared its performance to that of an optimal topology

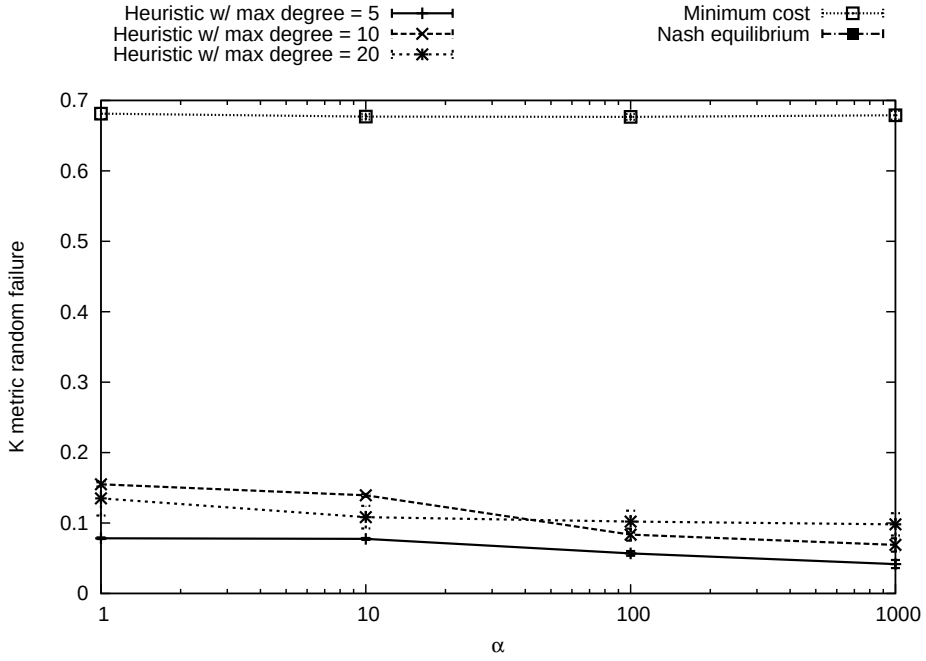


(a) 50 peers

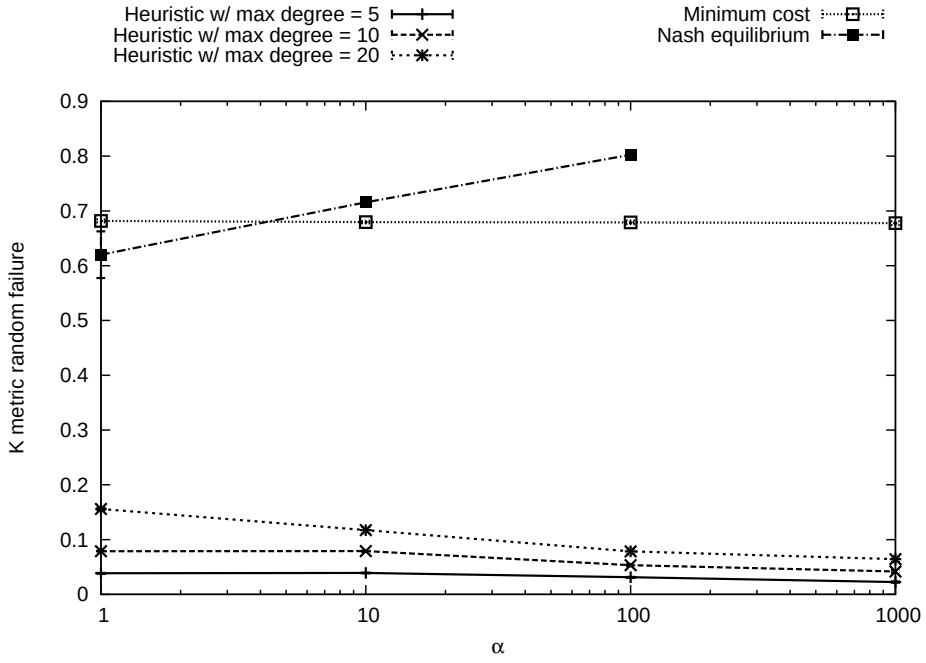


(b) 100 peers

Figure 4.7: K metric normal

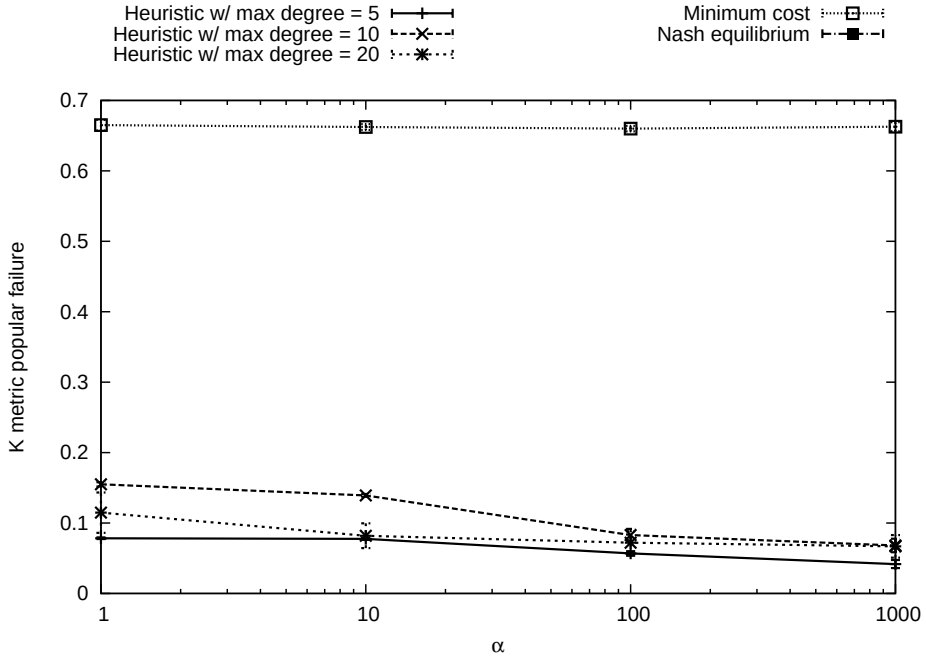


(a) 50 peers

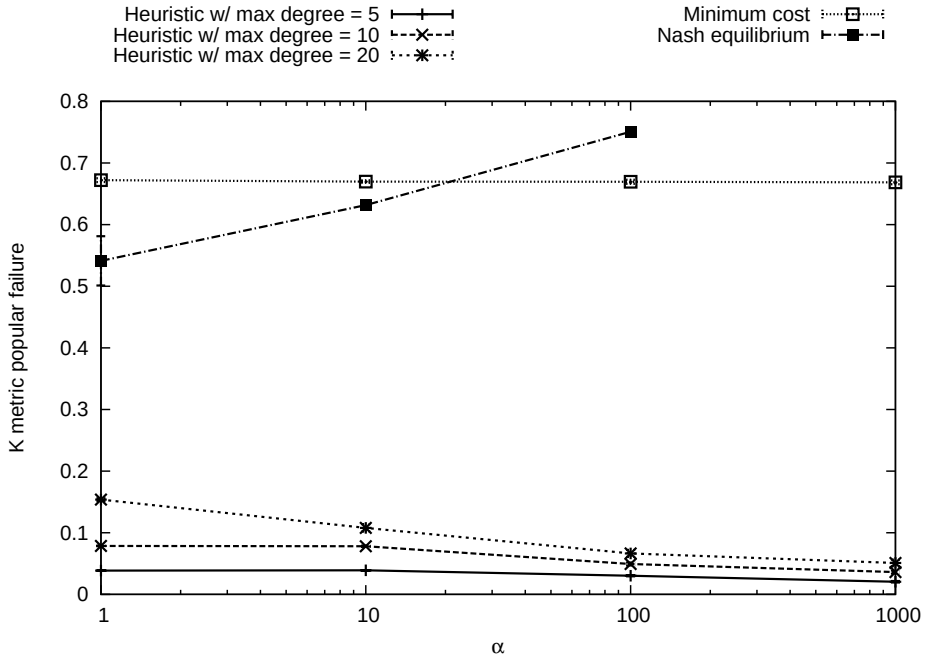


(b) 100 peers

Figure 4.8: K metric random failure



(a) 50 peers



(b) 100 peers

Figure 4.9: K metric popular failure

control algorithm that uses a game–theoretic model. The heuristic results were compared to the minimum cost results and the Nash equilibria of the game. The optimal algorithm is NP–hard to compute and, while serving as a useful benchmark, cannot be used for a practical P2P–MANET, necessitating the heuristic approach. Simulation results show that as the maximum degree restriction of the heuristic is relaxed, it approaches the minimum cost value. The next chapter provides a discussion of incentives in P2P–MANETs and also presents a server path selection algorithm.

Chapter 5

Incentive Mechanisms and Path Selection

A common problem in P2P file sharing networks are *freeloaders* or *free riders*, users that download files for themselves, but do not share anything in return. To entice users to share files, an incentive scheme is required. The general idea behind the scheme is to encourage users to contribute files by rewarding users who do so and possibly punishing those who do not. Selfish nodes are not exclusive to P2P systems. MANETs may also have nodes which do not forward traffic, thereby reducing the effectiveness and efficiency of the network as a whole. Such users also require an incentive, but one that encourages them to forward packets.

After a user has queried the P2P overlay for the data it is searching for, it is presented with a set of servers which have the file it is interested in. The MANET may provide multiple paths to each of these servers, resulting in a set of paths with different costs that the user must choose from. Furthermore, modern P2P file sharing services allow users to obtain files from multiple servers simultaneously by dividing

the file into blocks, or chunks.

In this chapter we provide an incentive scheme for P2P–MANETs which considers the issue of determining prices for sharing and forwarding in the P2P network. MANET nodes that are not part of the P2P overlay will still participate in the incentive scheme. We also provide a path selection algorithm which uses linear programming to produce the optimal set of paths for clients to select. Finally, we provide a heuristic algorithm which also selects paths, but with reduced computational complexity, and compare it with the optimal results. The path selection algorithm may be used in conjunction with the incentive scheme, although this is not necessary.

The remainder of this chapter is organized as follows. Section 5.1 presents an outline of the proposed incentive mechanism and path selection algorithms, including the objectives they are designed to meet. Section 5.2 presents the system model. Section 5.3 describes the incentive scheme in detail, while Section 5.4 describes both the optimal and heuristic path selection algorithms in detail. Section 5.5 provide a performance evaluation of the incentive scheme, and compares the path selection heuristic to the optimal solution obtained with the linear program. Section 5.6 summarizes the chapter.

5.1 Schemes' Outlines and Objectives

We propose an incentive scheme designed to encourage P2P overlay users to provide services such as sharing files and encourage MANET nodes to forward data. We also propose a path selection algorithm which, given a set of paths to servers and the associated costs, determines how many blocks to download from which servers and along which paths in order to minimize the cost, subject to a maximum download

time, or vice versa. Our schemes are designed to meet the following objectives:

1. Require users to contribute to the P2P overlay in order to obtain services
2. Allow nodes to price services to compensate them for the opportunity cost
3. Select the set of servers and paths to download from in order to minimize the cost or download time

An effective incentive scheme for P2P–MANETs encourages users to forward traffic and also, for those participating in an overlay such as a file sharing network, to share files. We assume a system in which the P2P overlay uses a system of virtual credits to enforce incentives. We focus on file sharing overlays in this chapter to simplify our description. When peers upload a file or when nodes forward data, they are awarded credits for doing so. These credits can then be used by the peers when they wish to download files themselves. MANET nodes which are not part of the overlay still receive credits for forwarding data, which they can then use should they later join the file sharing network. This allows nodes to be compensated for the cost of performing a service to the network, but requires them to be aware of the overlays. The credits are valid across different P2P overlays.

When a node forwards data or makes a file available for download, it is using its own resources, including CPU, bandwidth, and energy. The opportunity cost of forwarding data for other nodes is the use of one's own resources for the benefit of others because these resources are then unavailable for the node to use for its own purposes. Therefore, when a node considers its resources to be very precious, it should be able to price them accordingly. Specifically, if a node is low on energy, it may wish

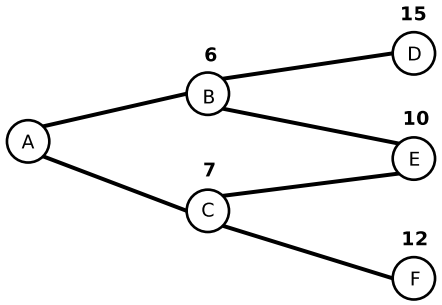


Figure 5.1: A choice of download paths and servers

to forward as little traffic as possible and thus charge a high price for this service. In contrast, if the node has abundant energy it can afford to charge a lower amount.

Given a set of paths consisting of servers that are willing to upload a file, as well as the total cost and estimated download time along a path to the server, it can be determined how much to download from each path in order to minimize the download time, subject to a maximum cost. This allows users to download from multiple servers simultaneously, resulting in faster downloads. Conversely, the user may elect to obtain the file at minimum cost, subject to a maximum download time.

Figure 5.1 shows client node A, which wants to download a file and has the choice of three possible servers. For simplicity, download time estimates have been omitted. Node A may download from server D via node B for a total cost of 21, from server E via B for a total cost of 16, from server E via C for a total cost of 17, or from server F via C for a total cost of 19. If A is concerned only with the cost, then choosing E via B is the cheapest. However, A might also wish to reduce the download time, and selecting multiple paths subject to a maximum cost and downloading simultaneously from them would accomplish this.

5.2 System Model

We assume the existence of a P2P file–sharing overlay in the P2P–MANET. The details of looking for the file and retrieving the results is left to a lookup protocol, which in unstructured overlays is generally some type of flooding technique, such as in Gnutella. We require that the full path to the server peer be specified in the responses, and the necessary virtual credits and estimated download time to also be given.

The server peer s_i will respond to the requesting client c's query with the three–tuple $\{s_i, p_{s_i}, d_{s_i}\}$, where p_{s_i} is the price that s_i will charge, and d_{s_i} is the estimated delay. These values may be for the entire file, or per block, depending on the content distribution system involved. This response is sent to the next hop on the path back to c. Each intermediate node t_i will determine its own price and delay and append the three–tuple $\{t_i, p_{t_i}, d_{t_i}\}$ to the response.

When c receives the response, it contains the entire path to the server, as well as pricing and delay information. c will then add all the price components, $P^l_{s_i} = p_{s_i} + \sum_{j=0}^{n-1} p_{t_j}$ to obtain the total cost for the file from s_i on path l. The total delay $D^l_{s_i} = d_{s_i} + \sum_{j=0}^{n-1} d_{t_j}$ is also determined. $P^l_{s_i}$ and $D^l_{s_i}$ are the sum of prices and delays for the i^{th} server on the l^{th} path respectively. c may obtain multiple responses from the same server s_i, that arrive via different routes. In that case, the intermediate nodes t_i on some of these l paths will be different, even though the server s_i remains the same. This results in different total prices and delay values. The totals, $P^l_{s_i}$ and $D^l_{s_i}$ are used to decide which server(s) and path(s) to select to download the file from.

We assume that there are a total of m download requests, each of which corresponds to one file request using the P2P overlay. These requests cover many different

files for many different clients. If a client wishes to download more than one file, than it will have more than one of the m requests. Each file is split into a set of blocks. We assume that the file associated with request i is split into B_i blocks, and each of these blocks is indistinguishable. In order to download the entire file, the client must acquire all B_i blocks. Clients set a maximum price they are willing to pay for each request, P_i, or a maximum time they must download the file within, T_i. The price may represent anything, such as hop distance, or the cost as determined in Section 5.3.

There are n paths which constitute all possible routes that connect the set of clients to the set of servers. Only one server is associated with each path, but there may be multiple paths per server. For a particular request i, and a path j, there is a cost per block $c_{i,j}$ and an estimated download time per block $d_{i,j}$. The server associated with path j possesses b_j blocks of the file. If $b_j = B_i$ then this indicates that the server has the entire file available to share.

The terminal node in the path is the server, which possesses the file to be downloaded. If the server associated with path j does not possess any blocks of the file corresponding to request i, it sets the cost and download time to infinity (i.e. $c_{i,j} = \infty$ and $d_{i,j} = \infty$) and sets b_j to zero.

5.3 A Utility–Based Incentive Mechanism

A P2P–MANET incentive scheme must promote sharing of resources, including traffic forwarding and file sharing. The incentive scheme proposed here, to our knowledge the first proposed for these networks, attempts to do just that. MANET nodes that forward packets are rewarded with virtual credits for doing so and peers also earn

credits for sharing files. Nodes that are part of the MANET but not currently part of the overlay may use the credits they have earned if they later decide to join the file sharing system.

In addition to the credit–based scheme, we introduce the idea of having each node indicate the delay it expects for the file transfer, so that a client can sum these up to obtain an idea of the total delay required to obtain a file via a particular route. The client can then consider both the credit cost and delay of obtaining the file from its peers in order to choose the best option for itself.

The technique used by the incentive scheme presented in this chapter consists of a client flooding the overlay with its file query, as in Gnutella. Servers that are willing to upload then send a response back to the client, including the cost of downloading the file, and also an estimated transmission time, accounting for files already on the P2P upload queue. Each intermediate node on the path back to the client adds in the price of its services, and also an estimate of the time to transmit the file. Results are then collected by the client, and after a certain waiting period, the choice is made.

It is important that the incentive scheme account for energy consumption, a vital consideration in MANETs. A P2P–MANET node should be able to decide to stop forwarding traffic because its battery level has fallen below a threshold and it wishes to conserve energy, or a node might decide that it will forward traffic only if it can charge a high price to compensate it for using highly limited bandwidth. Therefore, a P2P-MANET incentive protocol must consider a given device's capacity to serve the network, and allow it to establish a price that makes up for its costs.

The price charged should reflect the demand on a node to forward data. When a node is being asked to forward a lot of data, this may quickly use up its resources.

Therefore, it should charge more for them even if there is plenty left. This helps to prevent routes from becoming congested because central and popular routes will become overpriced and outlying routes will be more cost-effective. This also has the effect of being fairer to boundary nodes, who may not otherwise be able to earn credits for forwarding traffic.

It is undesirable for a node which has limited resources, such as little energy available, not enough bandwidth, or few spare CPU cycles, to forward traffic. Therefore the price that a node charges should also consider its available resources.

Initially, when a peer joins the network it starts with a small number of credits proportional to the number of files it is sharing, and is not allowed to let this number fall below zero. Basing the starting credits on how many files the user shares encourages sharing and discourages freeloaders, while still allowing new nodes to begin downloading files.

Suppose client c wants to obtain a particular file. It generates a query for that file (a *Query* message in Gnutella) and sends it to the peers it is connected to. This query is spread through the network by neighbouring peers.

When a server receives the request and has the desired file it must determine a price for it. The price of the file is based on two factors: its perceived popularity, and its perceived availability. The more popular a file is the more that can be charged for it. However, the more available a file is within the network the less its price should be in order to entice customers. File popularity is determined by the number of queries received for it in the past (the more queries received the more popular), while file availability is determined by the number of downloads in the past (the more downloads the less available it is). For example, if a peer has received many requests

for a file but few downloads, its offering may be too expensive relative to other servers, so it may choose to lower its price to encourage more downloads and thus earn more credits for itself.

The price charged by server s for file i is determined by the Cobb–Douglas [82] utility function

$$P_i^s = o_i^\gamma \times a_i^{1-\gamma} \tag{5.1}$$

where o_i is the perceived popularity of file i, calculated as an exponential moving average (EMA) of the number of requests for i, a_i is the perceived availability of i, calculated as an EMA of the number of downloads of i per request, and γ is a parameter that indicates s's preference for popularity against availability.

In addition to the price, s also calculates an estimated delay for serving the file. This delay can be determined in several ways such as based on historical information, or s can evaluate its outgoing queue size and bandwidth to calculate a delay value. We define the total nodal delay is a combination of three factors

$$d_s = \tau + T_x + D_e \tag{5.2}$$

where τ is the wait time in the server's file sharing upload queue including service time, T_x is the file transmission time, and D_e is the delay due to energy constraints. This equation takes into consideration many of the peer's resources, namely CPU utilization, queue length, bandwidth, and energy.

In this chapter, since we have no historical information on which to determine arrival and service rates, without loss of generality, we assume constants for these

values and thus use an M/M/1 queue. Therefore

$$\tau = \frac{1/\mu}{1-\rho} \tag{5.3}$$

where μ is the service rate of download requests and ρ is the utilization, equal to $\frac{\lambda}{\mu}$ where λ is the arrival rate of download requests. T_x is equal to the file size divided by the bandwidth.

The energy available at the server is an important consideration. If the node has little energy, it would be better off conserving it and not responding to requests. Therefore it indicates this preference by increasing the delay as energy decreases. D_e is defined as

$$D_e = \frac{k}{E+a} \tag{5.4}$$

in the range shown in Figure 5.2 where E is the energy available at the time of the request. In order to operate within the range shown in the figure, we define $a = \frac{D_{max} \times E_{min}}{E_{max} - E_{min}}$ and $k = \frac{E_{max} \times D_{max} \times E_{min}}{E_{max} - E_{min}}$, where D_{max} is the maximum delay, and E_{max} and E_{min} are the maximum and minimum energy levels, respectively. As seen in Figure 5.2, when there is very little energy available, delay is very high, but this decreases quickly as energy increases. When energy is at a maximum, the added delay is zero. A query response message (*QueryHit* in Gnutella) is sent back to client c, including the price and delay from the server.

In order for the client to obtain multiple results via different paths from the same server, the server must respond to duplicate requests, and realize that they are for the same request from the same client. In the process, the values of d_s, o_i, and a_i should be the same. The prices sought by the intermediate nodes are what differentiate the

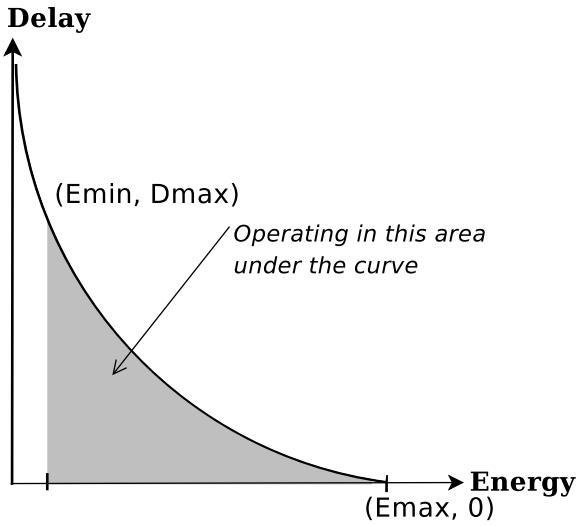


Figure 5.2: Representation of D_e

responses.

Each MANET node along the path back to c calculates its price of forwarding and estimated delay and appends this information to the response. It is in a node's interest to charge as much as it can in order to profit from providing the forwarding service. It is also in a node's interest to increase the delay as much as possible so that fewer resources are provided for other nodes, leaving more of its own resources for itself. However, the node must also be cautious not to set too high a price or delay, or the route will not be selected and the node will receive nothing. The only role a MANET node can play in the process is an intermediate node, since it is not participating in the overlay.

An intermediate node will determine its estimated delay using the function

$$d_i = T_x + D_e \tag{5.5}$$

Files are not placed on the intermediate peer's file sharing upload queue, and it is assumed that packet queues introduce negligible delay so the τ component discussed previously is unused here. The file size is transmitted as part of the query response, so the intermediate node can calculate T_x from this value, and D_e is calculated as before. Intermediate nodes, whether overlay members or not, do not cache the file; they simply forward data.

The intermediate node determines its price for forwarding the data based on its perception of the demand for its services. If it has been asked to forward a lot of traffic in the recent past, then the chances are higher than it will be selected again. Servers do not need to add this extra profit component as the price they charge already takes demand into account in the form of popularity and availability. The price that a node charges for forwarding traffic is the perceived demand calculated as an EMA of the amount of data forwarded,

$$p_i = \beta x_i + (1 - \beta) p_{i-1} \tag{5.6}$$

where x_i is the demand as of the ith packet, p_{i-1} is the historical price of the demand, and β is the smoothing constant.

When the query results reach c, they will contain the complete cost and delay of obtaining the file from that route, listed as a series of nodes with their associated prices and delay components from which c can calculate the values for $P^l_{s_i}$ and $D^l_{s_i}$. c can then choose the server(s) which satisfy its criteria, as discussed in Section 5.4. Additionally c must possess enough credits to obtain the file from the chosen server(s).

Suppose c elects to download the entire file from s_1 and it has sufficient credits. Then c would send a message to s_1 indicating its interest in obtaining the file, along

with the required credits and the list of nodes and prices on the path. s_1 would remove its price from the total amount of credits sent by c, adding this to its own credits. Next, it would enqueue the file on its upload queue, associating the request from c with it in order to keep the credit information intact. When the file request reaches the head of the queue, it is sent to the next hop along with all remaining credit associated with the query. Each intermediate node along the path would then take the credit due it, and eventually the file will reach c with no remaining credit. Pseudo-code for the actions of the servers and the intermediate nodes are given in Algorithms 6 and 7.

Algorithm 6 Server actions

function *receive-query*:
if not sharing file **then**
    ignore query
else
    update the values of o_i and a_i
    calculate price of uploading file based on popularity and availability, $P_i^s \leftarrow o_i^\gamma \times a_i^{1-\gamma}$
    estimate download queue delay, $\tau \leftarrow \frac{1}{\mu}/(1-\rho)$
    determine file transmission delay, $T_x \leftarrow$ file size / bandwidth
    estimate equivalent delay due to energy constraints, $D_e \leftarrow k/(E+a)$
    send response with price and delay information to next hop en route to client
end if

function *receive-dl-request*:
place request on queue
take credits

There may be nodes belonging to the underlying MANET which are not currently participating in the P2P overlay network. They may have been part of it in the past and/or may wish to join in the future. It is important that such nodes also obtain credit for forwarding traffic. Therefore, these nodes also determine price and delay

Algorithm 7 Intermediate node actions

1: function *receive-query-response*:
2: calculate price of forwarding file based on traffic demand, $p_i \leftarrow \beta x_i + (1 - \beta)p_{i-1}$
3: determine file transmission delay, $T_x \leftarrow$ filesize/bandwidth
4: estimate equivalent delay due to energy constraints, $D_e \leftarrow k/(E + a)$
5: forward query-response with new price and delay information to next hop en route to client

6: function *receive-file-transfer*:
7: take credits
8: send file to next hop en route to client

information while forwarding queries responses. This requires a modification of the underlying routing protocol.

The message types sent by the client are query and download request messages. The former indicates a search query that is propagated through the overlay, while the latter are sent to specific servers to indicate that the client wishes to download from them. These messages may include a specific set of blocks that the client wishes to download from that server. Servers send response messages to the query if there is a match. Intermediate nodes receive queries, which they propagate; query reponses, which they add to; and file transfers, from which they take their due and forward on to the destination.

The number of messages sent by the client node depends on the lookup protocol being used by the P2P–MANET. In the worst case, a flooding algorithm will propagate the client's query to all other overlay members, requiring $O(n - 1)$ messages to be sent. The number of responses would also be $O(n - 1)$ in the worst case, when all other peers respond.

The proposed incentive scheme works in both structured and unstructured overlays. In a structured overlay, such as Pastry [75] or Chord [78], once the client contacts

the index node and obtains a list of servers, it then contacts each server simultaneously to request price and delay information for the file. As the response makes its way back to the client, the intermediate nodes can calculate their price and delay and append it to the response. The client is then able to use this information to determining which server(s) to choose. In an unstructured overlay, the system works as described above.

5.4 Path Selection

In this section, we formulate the path selection problem as a mixed integer linear program (MILP) in order to determine an optimal solution such that the overall total download time is minimized and the cost of downloading the file does not exceed an amount specified by the client. The variables for the download times and costs can easily be interchanged to obtain the solution to the problem of minimizing the cost while not exceeding a maximum download time specified by the client. The linear program also ensures that no more blocks are requested from a server than it has available.

Next, we propose a heuristic approach which better supports the dynamic, mobile nature of the P2P–MANET. Clients select which peers to download from using a greedy algorithm. The client chooses the path with the best utility value and tries to obtain all the blocks available from that path. It then continues selecting paths with the next best utility value in a greedy fashion, until all blocks of the file have been downloaded.

5.4.1 Optimal Path Selection

We now address the problem of a group of clients selecting from a set of download paths, each with an associated cost and download time. The clients must determine not only which of the paths to download from, but also how much to download from each path. The files are assumed to be split up into a set of fixed sized blocks, and the clients determine how many blocks, if any, to download from each path. We assume that because peers download files in blocks, once they have at least one block of a particular file, they are able to share it. Therefore, the servers that clients may choose from do not necessarily possess the entire file. For simplicity, each block is assumed to be equivalent, something that can be accomplished using network coding, as shown in Chapter 6. It is also possible for a client to download blocks from the same server via multiple paths, using a multipath routing protocol such as DSR [46] or [80], or by forwarding at the overlay layer.

The problem to be solved is, given the set of n paths with known costs and times, determine for each request i, which of these n paths to select and how many blocks to download from each path. The goal may be either to minimize the overall download time subject to a budget constraint, or to minimize the cost subject to a time constraint. This problem is formulated as a mixed integer linear program. Let $x_{i,j}$ be an integer decision variable that indicates how many blocks the request i should download via path j. If $x_{i,j} = 0$, then no blocks of request i are to be downloaded from path j.

We first provide the MILP to minimize the download time, which is the number of blocks requested multiplied by the download time per block for each path selected. The download time problem is formulated as follows:

$$\text{Minimize} \qquad \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_{i,j} d_{i,j} \tag{5.7}$$

s.t.

$$\sum_{j=0}^{n-1} x_{i,j} c_{i,j} \leq P_i \qquad \forall i \tag{5.8}$$

$$\sum_{j=0}^{n-1} x_{i,j} = B_i \qquad \forall i \tag{5.9}$$

$$0 \leq x_{i,j} \leq b_j \qquad \forall i, j \tag{5.10}$$

$$x_{i,j} \in \mathbb{Z} \tag{5.11}$$

The counterpart cost problem minimizes the download cost and is formulated as follows:

$$\text{Minimize} \qquad \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_{i,j} c_{i,j} \tag{5.12}$$

s.t.

$$\sum_{j=0}^{n-1} x_{i,j} d_{i,j} \leq T_i \qquad \forall i \tag{5.13}$$

$$\sum_{j=0}^{n-1} x_{i,j} = B_i \qquad \forall i \tag{5.14}$$

$$0 \leq x_{i,j} \leq b_j \qquad \forall i, j \tag{5.15}$$

$$x_{i,j} \in \mathbb{Z} \tag{5.16}$$

These two programs differ only in what they are trying to minimize (Eqns. 5.7 and 5.12) and the first constraint (Eqns. 5.8 and 5.13). The former minimizes the download time and the constraint ensures that the maximum budget, P_i is not exceeded. The latter minimizes the cost of downloading and the constraint stipulates that the download time not exceed the maximum time, T_i. The next constraint (Eqns. 5.9 and 5.14) ensures that all blocks of the file are downloaded. The penultimate constraint (Eqns. 5.10 and 5.15) prevents non-negative block requests and also prevents downloading more blocks than are available on a given path. Finally, the last constraint (Eqns. 5.11 and 5.16) ensures that number of requested blocks is an integer.

This program finds the optimal number of blocks to request from each path, but clearly requires *a priori* knowledge of all client download requests, server locations, costs, and download times. In a dynamic P2P–MANET, this knowledge is not known beforehand, and so a heuristic algorithm is proposed in the next section. However, the linear program solution serves as a useful benchmark for the heuristic algorithm.

5.4.2 Heuristic Path Selection

We now propose a heuristic algorithm that each client executes on its own to select the paths. First, the client must query the network for the file it is interested in. We assume that there are multiple servers and paths available to select from. Each path has associated with it a total cost per block, an estimated download time per block, and the number of blocks the server associated with that path has available. The algorithm given in Section 5.3 can be used for this purpose, although any other

algorithm may also be used. The client must then determine how many blocks to download from each path, with the goals of having the lowest download time and cost, given the user's preference, that does not exceed the client's budget, and does not download more blocks than are available from a particular server.

For each path p_j, the client calculates its utility using the Cobb–Douglas form equation

$$U_{p_j} = c_{p_j}^{\delta} \times d_{p_j}^{1-\delta} \tag{5.17}$$

c_{p_j} is the total cost of each block on download path p_j, d_{p_j} is the download time of each block on path p_j, and δ indicates the client's preference of price to download time. With the heuristic algorithm, users can balance their preferences for download time and cost, unlike with the MILP, in which only one goal is optimized. A value of $\delta = 0$ prefers the lowest download time regardless of the cost, while $\delta = 1$ indicates minimum price, no matter the download time. δ values between 0 and 1 allow the user to attempt a balance of both.

The client now sorts these utility values in descending order and uses a greedy algorithm to download as many blocks as are available from the top paths. That is, the maximum number of blocks that are available from the path with the highest utility are downloaded. If more blocks are needed to complete the file, the path with the second highest utility is selected and the lesser of the number of blocks it has and the number of blocks remaining are requested from this second path. The algorithm continues in this fashion until all blocks are downloaded.

If the total cost of obtaining the file exceeds the client's budget, then the client has given a lower preference to the price than it can afford, and so it must correct this and try again. The client must increase the value of γ and recalculate all utility

values again.

5.5 Performance Evaluation

In this section we evaluate the performance of our proposed incentive mechanism, and also compare the performance of the heuristic path selection algorithm to the MILP path selection algorithm. The incentive mechanism and the path selection heuristic were implemented in the network simulator *ns–2* 2.33 [68]. The download requests and server paths obtained in the simulations were then used to create input files for the MILP solver, ensuring consistency of the results. The mixed integer linear programs were solved using lp_solve 5.5.0.10, a free linear programming solver that uses the revised simplex and branch–and–bound methods. We now discuss the simulation model and performance metrics, followed by a discussion and comparison of the results.

5.5.1 Simulation Model

For consistency, the simulation model used in this chapter is the same as the one used in Chapter 3. All MANET nodes are mobile, and begin with an energy level of 100,000 Joules. As the simulation proceeds, the energy of the nodes changes according to Table 3.1. Specific information for the incentive scheme simulations, and the path selection simulations follows.

Incentive Scheme

For the incentive scheme, each peer makes a total of 50 queries for a different, random file. Files are placed randomly throughout the network with between 10% and 60% of nodes having a given file. File sizes range from 1 MB to 10 MB. An unstructured Gnutella–like overlay is used. The following values are used for the constants: D_{max} = 10 sec, ρ = query rate/avg. file transfer time, $\gamma = 0.5$, $\beta = 0.375$, and the epoch for the EMA is five minutes.

To test the incentive scheme proposed in this chapter, we require the client to select from among the responses and download the entire file from a single peer. We assume that when the query results reach client c, they will contain the complete cost and delay of obtaining the file from that route, listed as a series of nodes with their associated prices and delay components. c then chooses the server s_i which gives the best value for a utility function, the Cobb–Douglas form equation

$$U_{s_i} = p_{s_i}^{\eta} \times d_{s_i}^{1-\eta} \tag{5.18}$$

p_{s_i} is the sum of prices charged by all nodes en route from s_i to c, d_{s_i} is the sum of all delays en route from s_i to c, and η indicates the preference of c for price vs. delay. In each simulation experiment, the price–delay preference parameter, η in the utility function is varied. When set to zero, the effect is that the price factor is ignored and the path with lowest delay is selected; when set to one, the effect is that the delay factor is ignored and the lowest priced download is selected. In between, the two factors are given weighting in accordance with the parameter.

In addition, simulations for overlays in which no pricing scheme is used and one

which has a fixed price for downloads and forwarding packets were also performed in order to compare them with the flexible pricing scheme presented in this chapter. For the unpriced scheme, the download was obtained from the first node to respond to the query, which generally favours closer servers. For the fixed price scheme, files were obtained from the cheapest path, which also favours closer servers. Free riders were not explicitly placed into any system, but as the results show they nevertheless made an appearance in the unpriced scheme.

Path Selection Algorithm

For the path selection heuristic, the total number of requests made is varied between 100 and 1000. The requests are generated by a random peer for a random file. As previously, the files are placed randomly through the network. Block sizes range from 100 to 1000 blocks, and between 10% and 60% of peers have a given file. Once again, an unstructured Gnutella–like overlay is used. The cost and delay values for the blocks were obtained using the incentive mechanism presented in Section 5.3.

The δ value of the utility function was varied to determine its effect on the path selected. The file requests, costs, and download times obtained from the path selection heuristic simulations are input into the MILP, and the program is solved to produce the optimal values. The cost and time constraints used, P_i and T_i, are the best values obtained by the heuristic. No actual downloads took place in the simulation, so to better simulate a P2P file–sharing network, and to obtain useful cost values, constant bit rate (CBR) traffic of 1000 byte packets sent 100 times per second was sent between members of the P2P overlay. The traffic started and stopped at random times, between random peers.

5.5.2 Performance Metrics

The following performance metrics are used to evaluate the incentive scheme:

number of downloads The average number of downloads each peer was able to complete. The maximum possible number is 50.

max and min downloads The number of file downloads achieved by the peer with the most downloads and the peer with the fewest downloads. This helps show how fairly the downloads were spread throughout the network.

max and min uploads The number of files uploaded by the peer with the most uploads and the peer with the fewest uploads. This helps show how fairly the uploads were spread throughout the network.

Jain's fairness index This well–known fairness index is used to determine the fairness of file uploads among peers. It is calculated as follows

$$\text{fairness} = \frac{(\sum_{i=0}^{n-1} x_i)^2}{N \sum_{i=0}^{n-1} x_i^2} \tag{5.19}$$

where x_i is the number of files uploaded by the i^{th} peer.

delay The average file download time, including queueing time. Lower download times are preferred by users.

energy consumption The normalized energy consumption of peers.

The following metrics are used to evaluate the path selection scheme:

delay The total download time of all file downloads.

cost The total cost of all downloads.

5.5.3 Simulation Results

In this section, we show and discuss the simulations results for both the incentive scheme and the path selection algorithms discussed in this chapter. The simulation results obtained in all experiments in this thesis have a 95% confidence level based on 10 independent runs. The confidence intervals are indicated in the figures below.

Incentive Scheme

Each peer sends 50 file queries so this serves as the upper limit on the number of downloads. Figure 5.3 shows the average number of downloads achieved by the peers in the overlay. In general, the number of downloads increases with overlay size because more servers are available to serve the file. In the proposed incentive scheme it can be seen that as the preference for reducing delay increases, the peer is able to obtain more files because the downloads tend to come faster.

The fixed price scheme has the fewest downloads because the pricing system it uses does not change with a peer's condition in the network. Peers are forced to offer a set price, regardless of whether it is to their benefit or not. Because peers download from the lowest cost route, the queuing time may be quite high, resulting in the download not completing before the simulation ends. Also, high demand peers cannot charge higher prices, and thus they earn fewer credit, which prevents them from downloading more files. The unpriced scheme has highest number of average downloads because it will download the file from any node and requests are never rejected due to insufficient credits.

Figure 5.4 shows the number of downloads from the peer which has the highest count and the peer with the lowest count. The figure shows that as more servers

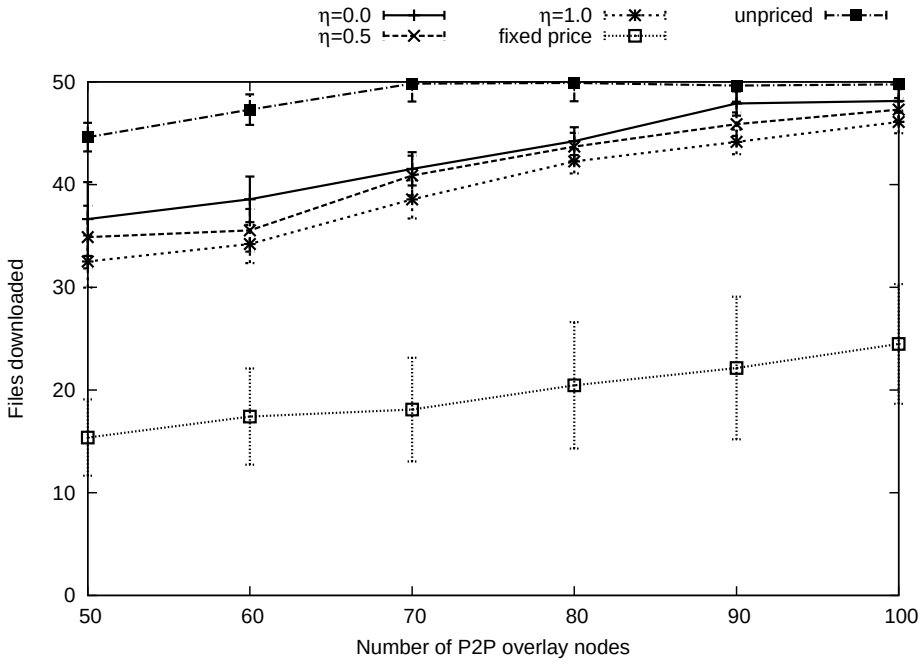


Figure 5.3: Average number of downloads per peer

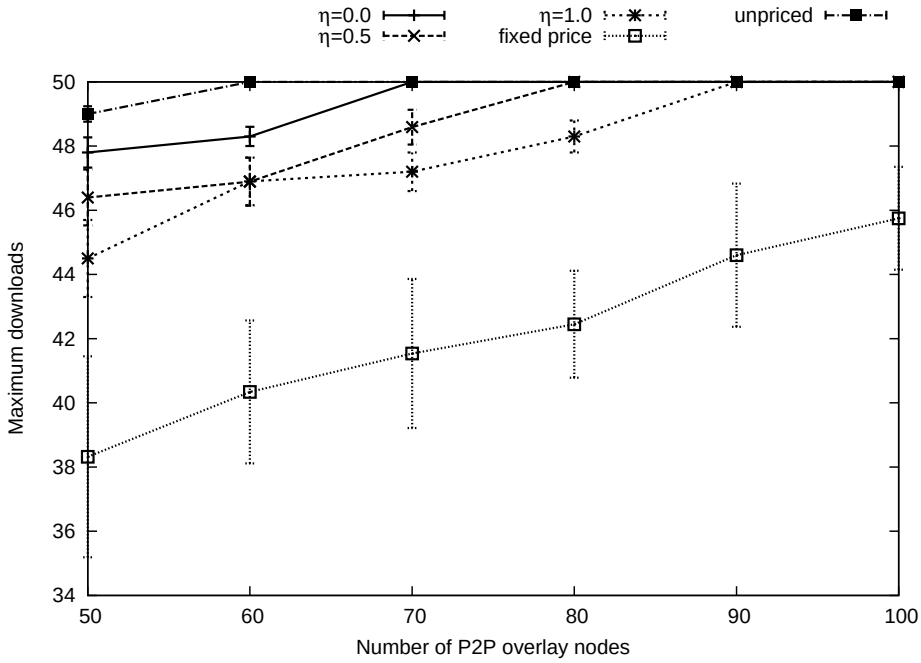
become available, peers tend to download more files. The unpriced scheme has highest maximum downloads and it is nearly always the maximum possible. It also has the highest minimum downloads for the same reason. The fixed price scheme has the fewest maximum downloads because sometimes nodes cannot afford to download the file they want because they do not have enough credits due to not serving enough files. It also has the fewest minimum downloads for the same reason. Some peers, likely those on the border of the MANET, are not able to get anyone to download from them since nothing differentiates their price from the others. Once they've spent their initial credits they can't download anymore. As the overlay size increases and more servers are available to upload files, the minimum download count declines since their chances of being a server decreases.

The proposed incentive scheme has a fairly high maximum download amount

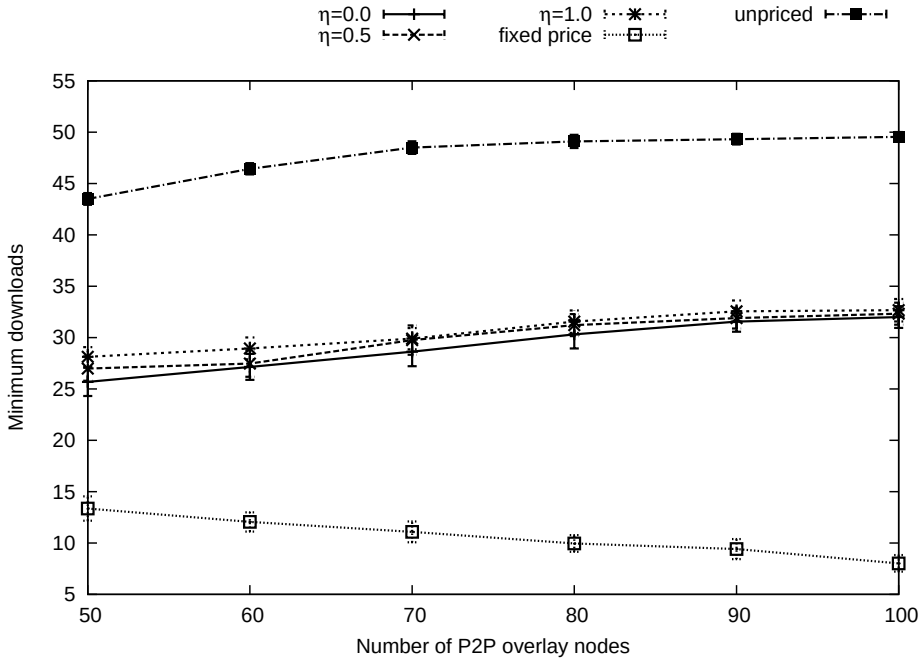
because peers can change their prices to attract peers or to compensate for heavy demand, thus giving them more credits and allowing them to download more. The minimum download rate is not as good as the unpriced scheme but is much better than the fixed price scheme. If peers don't have files others want, they can't do much about it other than to forward traffic and try to reduce their prices and make up for it with volume.

Figure 5.5 shows the number of uploads from the peer which has the highest count and the peer with the lowest count. As more servers become available, peers tend to upload less on average. The unpriced scheme has highest number of maximum uploads because some nodes, the centrally located peers, will be closest to the most nodes and thus upload lots of files. Others, the border nodes may not upload any files at all. Even though no peer ignores a request if it has the file, there are still freeloading nodes in the unpriced system. These are the border nodes who have nothing of interest to nearby nodes, and are too far to be the uploading node for others. The fixed price scheme also has some nodes with very few uploads and some with a very high number. Again, the border nodes tend to upload less, while the central nodes upload more. In this sense, the fixed price and unpriced schemes perform very similarly. The proposed scheme tends to more evenly spread out the uploads across nodes. If a central node finds that it is uploading a lot, its price goes up, and so downloading peers will find alternative sources and routes. This results in more fair sharing of the contribution burden within the overlay.

Figure 5.6 shows Jain's fairness index values for the number of files uploaded for each system over different P2P–MANET overlay sizes. Higher values indicate greater fairness, with perfect fairness achieved at a value of 1. The fairness of all systems

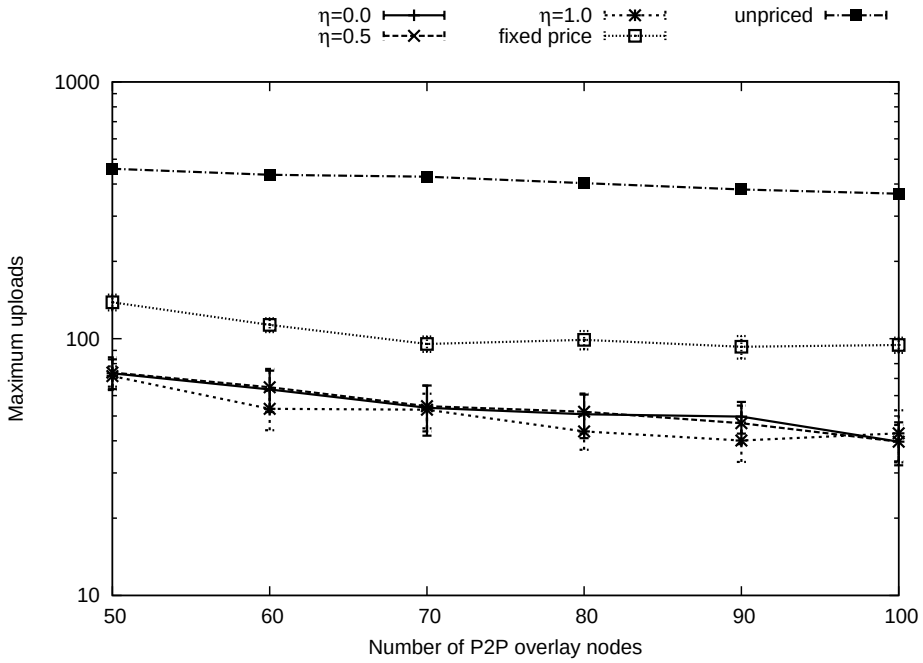


(a) Max downloads

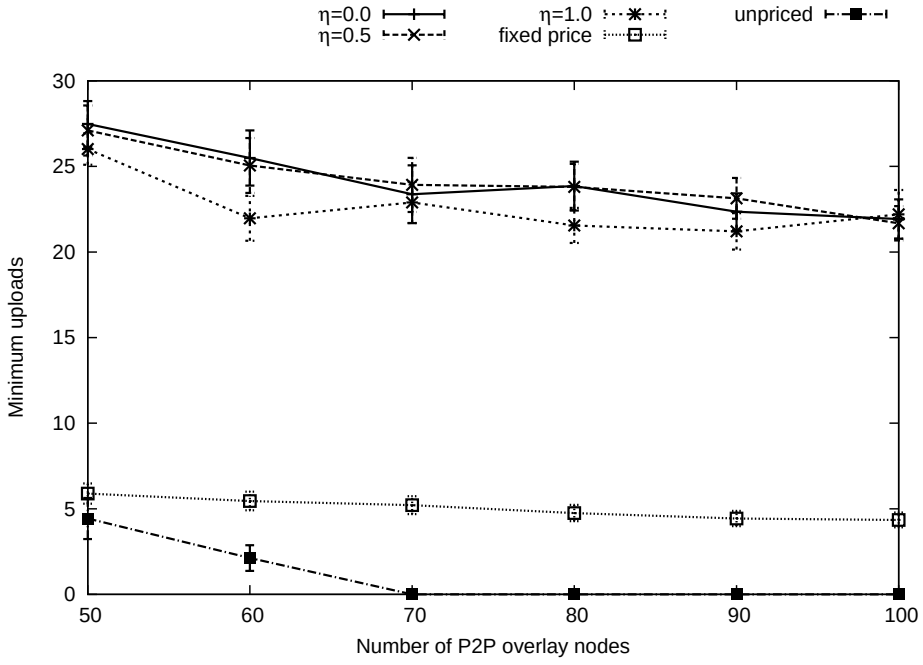


(b) Min downloads

Figure 5.4: Maximum and minimum downloads



(a) Max uploads



(b) Min uploads

Figure 5.5: Maximum and minimum uploads

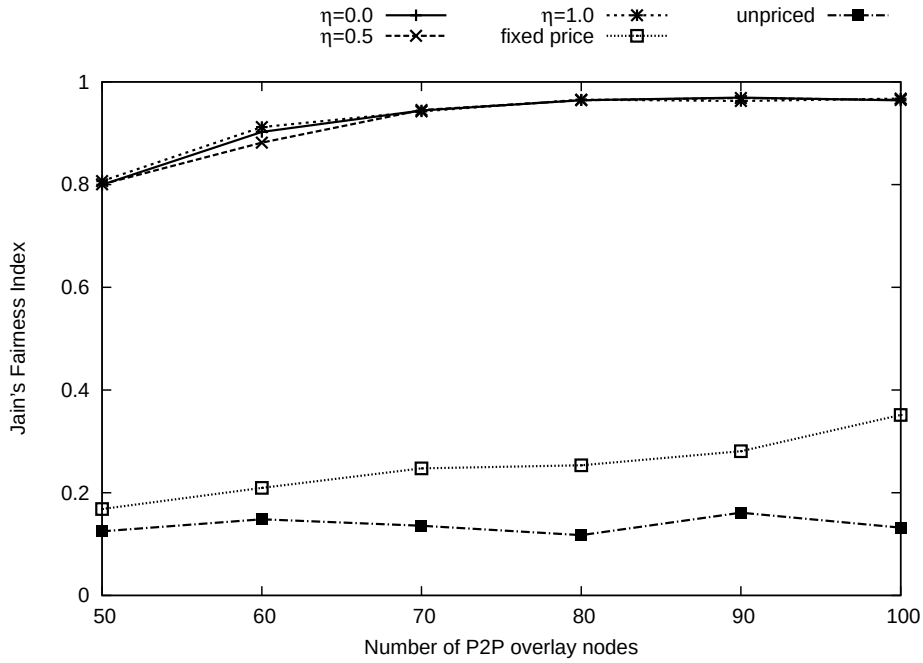


Figure 5.6: Jain's fairness index for file uploads

increases as the overlay size increases because there are more peers available to upload files since file placement is a percentage of the overlay size. However, the unpriced and fixed price schemes are significantly less fair than the proposed algorithm. As explained previously, the proposed algorithm allows peers to adjust their prices to attract or dissuade more downloaders, resulting in more even sharing. The fixed price and unpriced schemes have border peers participating much less in sharing than the central peers.

Figure 5.7 shows the average delay per download. It can been seen that this time is lowest for the proposed scheme, and particularly when the delay preference is set highest. This results in a peer always choosing the server with the lowest estimated delay. A high cost preference chooses the lowest cost path and ignores the delay, so it has much higher delay. The unpriced and fixed price schemes also ignore delay and

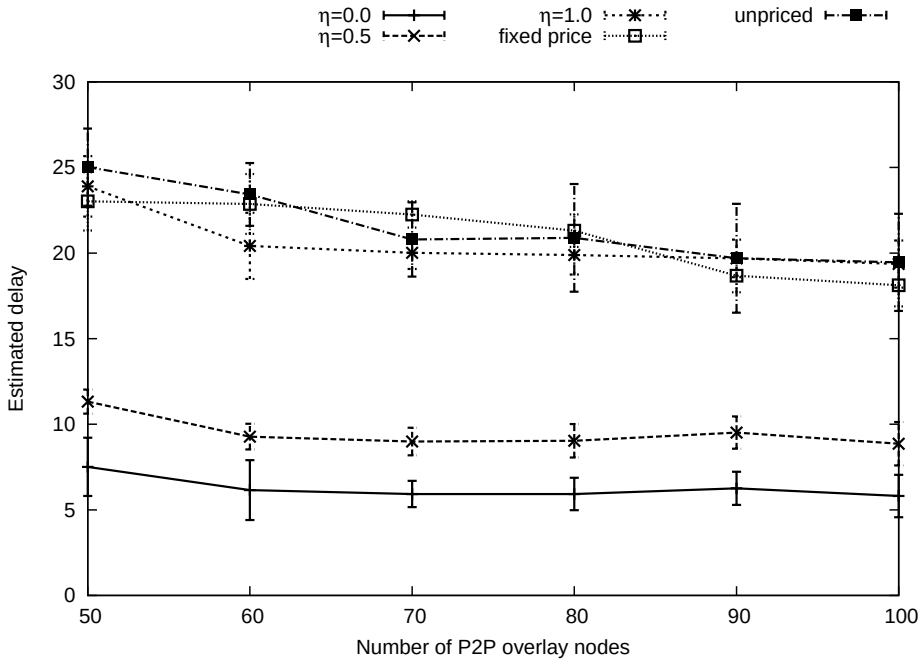


Figure 5.7: Average delay per download

choose the first peer to respond, however this peer may have a long queue, so the delay tends to be higher.

Figure 5.8 shows the normalized energy consumption of peers. The energy consumption is lowest for the fixed price scheme because it has the fewest downloads. It tends to rise with number of peers because the downloads tend to travel over longer distances as the overlay grows in size. The proposed scheme has a fairly constant energy consumption because even though nodes download more with increasing network size, the distance the download travels falls due to the dynamic pricing scheme for higher values of the delay preference. The cost–focused version has higher energy usage because the downloads generally must travel farther. The unpriced scheme has the highest energy consumption because it permits the highest number of downloads and those downloads tend to travel farther in the network as well.

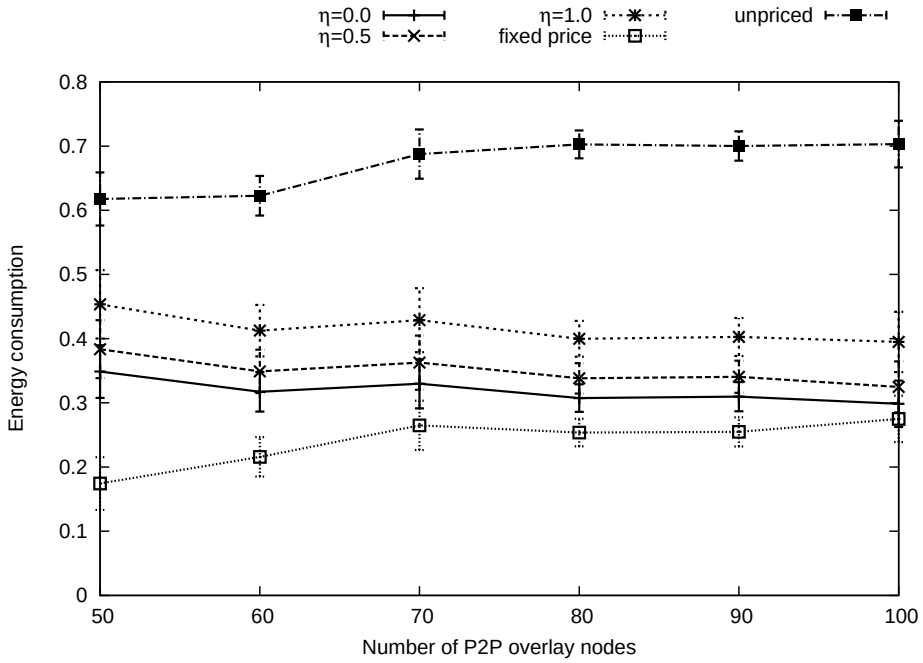


Figure 5.8: Normalized peer energy consumption

Path Selection Algorithms

Figure 5.9 shows the total delay over all file downloads. The linear program which optimizes the delay has the lowest delay, but the difference between it and the delay–focused version of the heuristic path selection algorithm is very small. This shows the relatively strong performance of the heuristic. As the focus of the heuristic shifts away from delay toward cost, the delay rises. The linear program that optimizes the cost has the largest delay, since this is not optimized in the program.

Figure 5.10 shows the total cost over all file downloads. The linear program which optimizes the cost has the lowest cost, but again, the difference between it and the cost–focused version of the heuristic path selection algorithm is small. Given that the heuristic does not require *a priori* knowledge of all file requests, locations, and costs, and its relative simplicity, it appears to have strong practical utility. As expected,

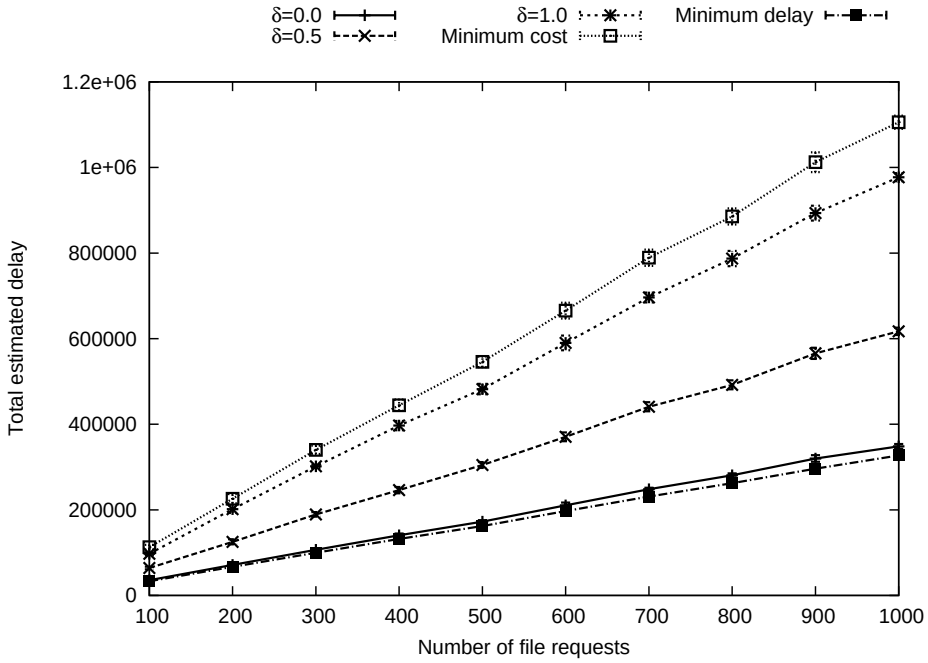


Figure 5.9: Total delay

when the focus of the heuristic shifts away from cost toward delay, the cost rises. The linear program that optimizes the delay has the highest cost, as in the previous case, since it is not optimized in the program.

The proposed incentive scheme has been implemented as a Java application and is presented in Appendix B.

5.6 Summary

This chapter provided a P2P–MANET incentive scheme, which eliminates the free–riding problem and increases the utility of the file sharing overlay. The performance of the scheme was shown to be superior to that of both unpriced and fixed price

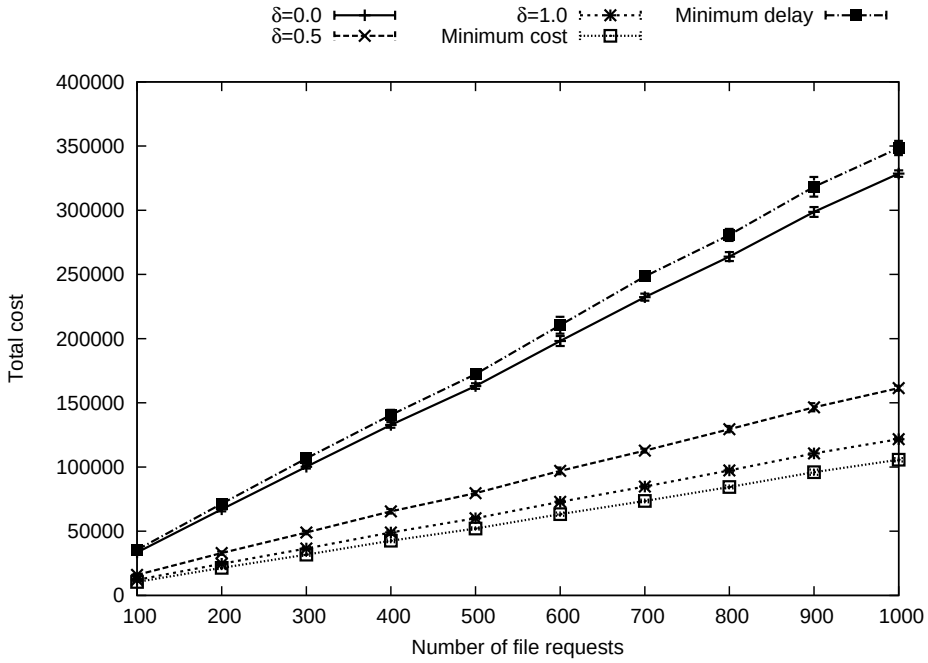


Figure 5.10: Total cost

overlays. Selecting download paths was also discussed, and MILP and heuristic algorithms were proposed. It was shown that the heuristic algorithm performed quite well as compared to the optimal MILP. An implementation of the incentive scheme operated successfully on a physical network, as shown in Appendix B. The next chapter presents an efficient content distribution scheme that allows peers to quickly download files while consuming less energy.

Chapter 6

Content Distribution

Content distribution is a means of transferring large amounts of data to interested parties. The use of P2P networks for content distribution has become more prevalent due to its high scalability. In a typical client/server model, all users acquire the data from a server. As the number of users attempting to download the content increases, the uplink bandwidth of the server becomes congested and the transfer speed to users drops, and the server may even become unavailable. In a P2P network, as more users join in order to download the content, they in turn share their uplink bandwidth. This enables parts of the data to be replicated throughout the network, and the system is better able to deal with a large number of simultaneous downloading users. P2P content distribution is therefore more resilient and provides greater availability due to the the replication of content throughout the network [55].

Many P2P networks, particularly file–sharing networks, exist to distribute large data files, or content. In a MANET, due to the non–existence of stationary, wall–powered servers, content that is distributed in P2P fashion is preferred over the traditional client/server model due to the ease with which a MANET node's bandwidth

can be overwhelmed and its limited energy. Therefore, a means by which content can be distributed that reduces bandwidth requirements and energy consumption is desirable. Furthermore, from the user's perspective, being able to download content more quickly is beneficial.

In this chapter we introduce a content distribution scheme that takes advantage of network coding and multipoint–to–multipoint communication to provide an efficient means of transferring files between peers in the network. Client peers locate server peers and download coded blocks, which enables them to retrieve content in less time than downloading uncoded blocks.

The remainder of this chapter is organized as follows. Section 6.1 outlines our proposed content distribution scheme and discusses its objectives. Section 6.2 describes the system model. Section 6.3 presents the content distribution scheme in detail. Section 6.4 includes a performance evaluation of the proposed content distribution scheme, comparing it with several alternatives. Section 6.5 summarizes the chapter.

6.1 Scheme Outline and Objectives

We propose an efficient content distribution system for peer–to–peer computing in mobile ad hoc networks, to our knowledge the first one. The proposed scheme takes advantage of linear network coding to eliminate the rarest–block problem and multicasting to reduce the number of transmissions where possible. This results in reduced energy consumption and decreases download time. Our scheme is designed to meet the following objectives:

1. Reduce energy consumption

2. Decrease download times

3. Efficiently support large file distribution

4. Eliminate the rarest–block problem

There are currently two primary techniques for distributing data in P2P networks. We refer to these as the whole–file and the multi–block techniques. In the whole–file case, a peer, after locating the content on the network, will attempt to acquire the data from another peer which possesses the desired content. We will refer to this node as a server. The client peer will download the entire content from a single server peer. This is similar to the traditional client/server model, except that there may be several servers within the P2P network, and the client peer, once it has downloaded the entire file, will itself be available as a server peer for other nodes.

The multi–block technique splits up the content into many fixed size blocks of data. A client peer may then download blocks from any server peer such that all blocks that make up the file are eventually downloaded. This allows the client peer to download the content from serveral server peers simultaneously. An advantage to this technique is that peers which have only partially downloaded the content may still act as server peers for other nodes, able to upload the blocks they possess. Peers with the entire file are referred to as seeds.

One issue with this technique is determining which blocks to download from which server peers. A common problem is the "rarest block" problem, in which the file block with the fewest number of replications network–wide may be difficult to acquire. Therefore, many multi–block algorithms try to acquire this block first in an attempt to "spread out" the copies of blocks through the network.

Network coding has been shown to improve performance in wireless environments [48, 49, 41, 29] as well as in P2P overlays [34, 35, 36]. In wireless networks, network coding allows nodes to combine packets corresponding to different streams, increasing the information content of each transfer and as a result, total throughput. In mobile networks, this also saves device resources, such as energy. In P2P overlays, network coding eliminates the rarest block problem altogether. By encoding across all available blocks, and then transmitting the encoded blocks, all blocks become equivalent, and it is simply a matter for the client peer to acquire enough linearly independent or "innovative" blocks.

Furthermore, in a MANET the use of multicasting becomes available, and so it is possible for a server peer to multicast its encoded blocks to many client peers. This allows a single transmission on its part to reach many client peers. Since the blocks are all identical, there is no need for client peers to locate and identify specific blocks. The requisite number of innovative blocks suffices. In turn, client peers may download from multiple server peers simultaneously, enabling an efficient multipoint–to–multipoint content distribution mechanism.

6.2 System Model

We consider a peer–to–peer mobile ad hoc network, in which the peers are already joined in the overlay and the topology of the network has been determined. Any bootstrap mechanism and topology control algorithm may be used.

The content to be distributed comes in the form of a fairly large file that a client peer wishes to download. We further assume that when a client peer wishes to download a file, it has already run a search query and possesses a list of servers that

have at least one encoded block constituting the file, and also the value of the cost function for each server. That is, the client peer wanting to download a particular file knows of all the server peers within the overlay, including how many encoded blocks each server peer has, and the cost to download each block from that server peer. How the client peer determines this information is part of the file query mechanism portion of the P2P network, not the content distribution part, and a system such as that proposed in Chapter 5 may be employed for that purpose.

For the purposes of this chapter, we assume the cost function used in the network is hop distance. Since client peers wish to minimize their cost, this means that a closer node is preferred to a farther one. In the event that some incentive system is in place, it may provide the value for the cost function. For example, if a credit–based incentive scheme is being used, the client peer could choose server peers with the lowest credit price.

The file to be distributed is split into k blocks, and clients must obtain at least k innovative blocks. The blocks sent by a server are a linear combination of all the blocks it currently has.

Assume the server has m blocks, where $m \leq k$. For each of the m blocks the server chooses m different, random coefficients, $\{g_1^1, \ldots, g_m^1\}, \ldots, \{g_1^m, \ldots, g_m^m\}$. All operations take place in a finite field of size 2^s. As explained in Section 2.6, even with a field size of 2^8 the probability of selecting linearly dependent combinations is negligible. In this chapter, we choose a field size of 2^{16}, i.e., $s = 16$.

To encode the file, every s bits of block i are multipled by the coefficient g_i^j, and all these components are added together to produce encoded block B^j, i.e., $B^j = \sum_{i=0}^{m-1} g_i^j \times i$. We define the $\times$ operator here to mean that the bits are XORed

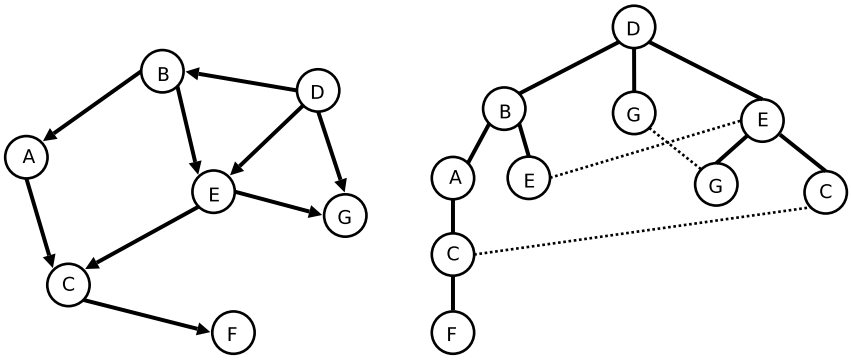
together, s bits at a time. The data sent to the client includes the coded block B^j as well as the coefficient vector, $\vec{g^j} = \{g_1^j, \ldots, g_m^j\}$.

To recover the original blocks, the client must obtain at least k linearly independent blocks. These blocks, along with the associated coefficient vectors are considered as a system of linear equations, and any way to solve for the unknowns, i.e., the original blocks, such as Gaussian elimination, can be used.

6.3 Efficient Multipoint–to–Multipoint Content Distribution Scheme

In a P2P file sharing system the bulk of the network traffic will consist of the files being transferred through the network. It is therefore important to manage the file downloads to make the transfer of content as efficient as possible, both in terms of reducing download times and reducing energy consumption. We now examine the problem of how to manage downloads in a completely decentralized fashion, while also trying to reduce download times and energy consumption.

There is no centralized authority and no infrastructure in a P2P–MANET, therefore a tracker node cannot be used. Instead we make use of multicasting for efficient communication within the overlay. The proposed algorithm uses the idea of a server node multicasting blocks, something that is less practical on the Internet, but more so in MANETs. Clients request a certain number of blocks from multiple servers depending on the cost of acquiring them, resulting in multipoint–to–multipoint communication. The determination of how many blocks to download may proceed as in section 5.4.2. We also propose the use of network coding to reduce file download



(a) Topology view (b) Tree view

Figure 6.1: Multipoint–to–multipoint content distribution

times. Encoding the file blocks will likely also have the effect of reducing the number of blocks that peers must download, which reduces energy consumption.

Figure 6.1 illustrates how such an approach would work. Node D is the seed node, meaning it has the entire file. Nodes B, E, and G are downloading directly from D, which is multicasting file blocks to them. Other nodes in turn are downloading from those nodes. Figure 6.1a shows a topology–based view of the downloads, with the arrows indicating the direction of transfer. Figure 6.1b shows the same information, but using a tree–based view for clarity. The dotted lines connect the same nodes, indicating their presence in multiple subtrees. As an example, Node E is receiving blocks from multiple servers, B and D, while simultaneously multicasting blocks to multiple receivers, C and G, illustrating the multipoint–to–multipoint communication concept.

We assume that the client has already run a search query and has determined how many blocks to download from a particular download path. Server nodes perform network coding on all of the blocks that they currently have, as described in Section 6.2. Because the blocks are encoded, all blocks have the same level of importance

and the client must simply download enough innovative blocks.

When a server receives a request for blocks, it sets up a multicast group and informs the requesting client of the multicast group address. If the server was already uploading blocks, it sends the address of the existing group. All blocks sent by the server are multicast. In this way, a single send can result in the block being received by multiple clients. Clients download blocks from multiple senders, and servers upload blocks to multiple receivers. The resulting multipoint–to–multipoint communication is expected to be very efficient, reducing energy costs and also download time. Any multicast routing protocol may be used.

6.3.1 Multipoint Download

Given the list of servers, block counts, and hop distances, the client uses a greedy algorithm to determine from whom to download, and how many blocks to request from each server. Because the blocks are encoded, all blocks have the same level of importance and the peer must simply download enough innovative blocks.

The steps taken by client c to download a file are shown in Algorithm 8. A counter, b, is initialized to k, the total number of blocks the file to be downloaded has been split into. Next, the list of servers is sorted based on cost, with the lowest cost server appearing first. The server at the top of the list, s, is then sent a download request. The number of blocks requested is the minimum of the number of blocks remaining to be downloaded, b, and the number of blocks the server has, n_s. This number is then deducted from the counter and if more blocks are needed, the next server on the list is contacted. The algorithm results in c requesting all of the blocks of the file from the servers such that the cost is minimized.

Algorithm 8 Client download algorithm

1: $b \leftarrow k$
2: sort list of servers in order of lowest cost
3: **repeat**
4: $s \leftarrow$ front of list
5: remove s from list
6: send request for $min(b, n_s)$ blocks to s
7: $b \leftarrow b - min(b, n_s)$
8: **until** $b = 0$

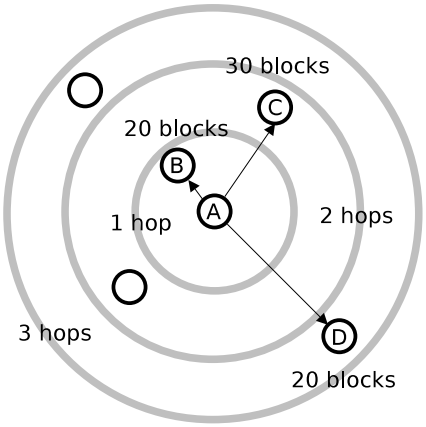


Figure 6.2: A client downloading blocks from the nearest servers

As mentioned earlier, the cost function used in this chapter is hop distance. Therefore, the closest server is contacted first and as many blocks as are available are downloaded from it. If more blocks are required, the next closest server is contacted, and so on. This is illustrated in Figure 6.2. In the figure, Node A is attempting to download a file which consists of 60 blocks. Node B, within 1 hop, is contacted first and 20 blocks are requested from it, since those are all the blocks B has. Node C, 2 hops away, is then contacted, and 30 blocks are requested from it. Finally, node D, which is 3 hops distant, is contacted. Only 10 blocks remain, and so 10 blocks are requested from D instead of the 20 available. The unlabeled nodes do not have any blocks of the file.

Once the client has obtained the k blocks of the file, it will attempt to solve the system of equations. If it has obtained at least k innovative blocks, then it will be able to obtain the original k unencoded blocks that constitute the file.

Clients choose which server peers to download from, and it is expected that most blocks will be linearly independent. However, it may happen that due to client, server, or intermediate node mobility, or the receipt of too many non–innovative blocks, a client may not be able to decode the entire file as expected. In this event, the client performs a new search query to get an updated list of servers and costs, and then re-runs Algorithm 8 but does not execute line 1. This allows the client to obtain the remaining number of blocks required.

The only message type sent by clients is the download request, which includes the number of blocks desired by the client. This message may be sent to multiple servers simultaneously.

6.3.2 Multipoint Upload

When a server s receives a request for blocks, it sets up a multicast group and informs the requesting client c of the address for the group. The details of this process depend on the multicast algorithm used. If s was already uploading blocks, it sends to c the address of the existing group. All blocks sent by s are sent via the multicast protocol and are encoded prior to sending. In this way, a single send can result in the block being received by multiple clients. Servers do not reject download requests because they could simply have ignored the search query if they did not intend to upload.

Algorithm 9 shows the steps s performs when it receives a download request. The incoming request from c contains the requested number of blocks, n. s keeps a

counter of blocks that have been requested to be sent, b. Since all the blocks it sends are encoded, the blocks themselves are indistinguishable. When a request comes in, b is incremented by n. The particulars of the creation of a multicast group and tree depend on the multicast protocol employed. All multicast protocols are compatible with the proposed algorithm.

Algorithm 9 Server actions

1: function *receive-dl-request*:
Require: An integer $n \geq 0$
2: $b \leftarrow b + n$
3: **if** existing multicast group **then**
4: send group address to client
5: **else**
6: create new multicast group
7: send group address to client
8: **end if**

9: function *upload-blocks*:
10: **repeat**
11: generate $\vec{g^j}$
12: encode block $B^j = \sum_{i=0}^{m-1} g_i^j \times i$
13: multicast B^j and $\vec{g^j}$ to clients
14: $b \leftarrow b - 1$
15: **until** $b = 0$
16: release multicast address

To send the data, s generates a random coefficient vector, $\vec{g^j}$, and then encodes the current coded block B^j. The encoded block and the vector are then multicast to the group address, and b is decremented. The process continues for as long as $b > 0$. It is important to note that the value of b will increase when a new client joins the multicast group.

Once s has finished submitting all blocks, i.e. $b = 0$, it ceases sending and makes the multicast group address available for use by others. Download clients, when they

have completed downloading the file, simply leave the multicast group. There is no need to inform the server that they have finished downloading, since the server keeps its own counter of blocks remaining to be sent.

6.4 Performance Evaluation

In this section we evaluate the performance of our proposed content distribution scheme using the network simulator *ns-2* 2.33 [68]. We begin by discussing the simulation model and then present the performance metrics that are used to evaluate the scheme. This is followed by a detailed discussion and comparison of the simulation results.

6.4.1 Simulation Model

We are attempting to determine the performance of distributing fairly large amounts of content. Therefore, we have selected two file sizes that each client peer is trying to download: 100 MB and 1 GB in size, split into either 1000 or 10,000 blocks each of size 100 KB. Each experiment runs for up to four simulation hours, sufficient time for all peers to download the file for all systems tested. Otherwise, the simulation model used in this chapter is the same as the one used in Chapter 3. MAODV [72] is used as the multicast routing protocol and AODV [71] is used as the unicast routing protocol.

All nodes are evenly distributed in the simulation area. The random waypoint model is used for mobility, with nodal velocities distributed according to a uniform distribution, with a minimum speed of 1 m/s and a maximum speed of 3 m/s. The

pause time is uniformly distributed, with a mean of 60 s.

The energy consumption model used in the simulations is the linear model proposed by Feeney [32] and used in earlier chapters. See Table 3.1 for the values of the constants. Nodes start with 100,000 Joules and leave the overlay when they've used 90% of their energy.

We compare our proposed scheme to three other systems. The first is a trivial content distribution scheme, in which a server uploads the entire file to a client, which we call the "whole–file" scheme. Client peers determine the lowest cost server peer, and attempt to obtain the entire file from it. The file is uploaded in blocks, to remain consistent with the other schemes, though all blocks must be obtained from the same server. Another popular scheme used in many P2P networks today is the "rarest first" multi–block scheme, in which client peers obtain the file's blocks from multiple server peers simultaneously. The server peer is selected according to whichever is the lowest cost at the moment of the block request. The client peer will attempt to obtain the "rarest" block first in order to try and evenly spread out the distribution of blocks. The third comparison scheme is one similar to Avalanche [34, 35, 36], which uses network coding, but not multicasting. In addition, there is no tracker available to coordinate peers. Client peers download encoded blocks simultaneously from multiple server peers according to whichever are the lowest cost. The cost function used is distance, so that nearer servers are preferred over more distant ones.

6.4.2 Simulation Parameters

The number of peers in the overlay varies between 50 and 100, in increments of 10. This allows us to examine the performance with both low and high numbers of peers.

The content consists of a single file, of size either 100 MB and 1 GB, depending on the experiment. The download interval time is 30 seconds, so a random node will start to download the file 30 s after the previous one began. This, combined with the varying number of peers, allows us to examine the performance under several different traffic intensity levels.

Experiments are run with the number of initial seeds varying as 1, 5, and 10. Seeds are peers that possess the entire file and the initial seeds are randomly placed in the network. As client peers begin downloading they may then provide blocks to other users, depending on the system being tested.

6.4.3 Performance Metrics

The following performance metrics are used in this chapter:

file download completion rate The average ratio of non–seed peers that complete the file download over the number of non–seed peers.

file download time The average time taken to download the file.

initial seed energy consumption The average amount of energy consumed by initial seeds. Each peer has a maximum of 100,000 J of energy, but leaves the overlay after using 90% of its capacity.

non–initial seed peer file blocks uploaded The average number of blocks uploaded by non–initial seed peers.

initial seed peer file blocks uploaded The average number of blocks uploaded by initial seed peers.

Jain's fairness index This fairness index is used to determine the fairness of block uploads by all peers, including seeds. It is calculated as shown in Section 5.5. Since not all peers have blocks to upload until later in the duration of the experiment, the maximum fairness will be less than 1.

6.4.4 Simulation Results

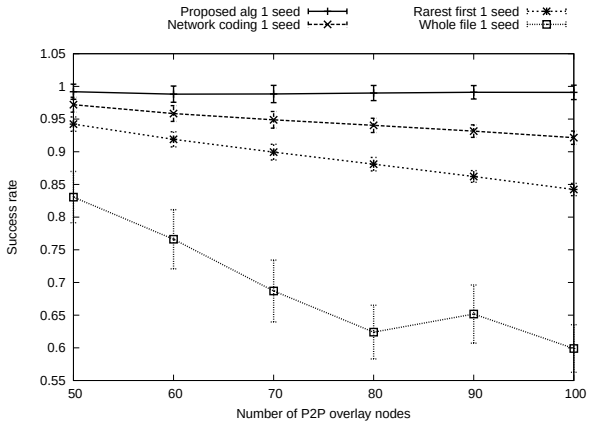
In this section, we present and discuss the simulation results for the numerous cases considered in our experiments. The simulation results obtained in all experiments in this chapter have a 95% confidence level based on 10 independent runs. The confidence intervals are indicated in the figures below via error bars.

The main goal of a content distribution scheme is to allow peers to obtain the content requested. Figure 6.3 shows the fraction of peers that were successful in doing so for the 100 MB file. The proposed algorithm had near–perfect success rates for all overlay sizes and seed counts. The use of network coding and multicasting allows essentially all peers to obtain the file. The scheme with network coding but no multicasting also has a fairly high success rate, though it tends to fall slightly as the overlay size increases due to the large increase in network traffic. Here, the proposed algorithm succeeds with a higher rate because its use of multicasting causes a slower increase in traffic as the overlay size increases. The rarest first block technique also performs reasonably well, with a decrease in success as the overlay size increases, also due to the increase in traffic. With more peers requesting the content, the network traffic causes congestion and dropped packets. For the whole–file scheme, the success rate starts off lower and falls more severely as the overlay size increases. Since all peers must obtain the file only from those that have the entire file, initially all peers

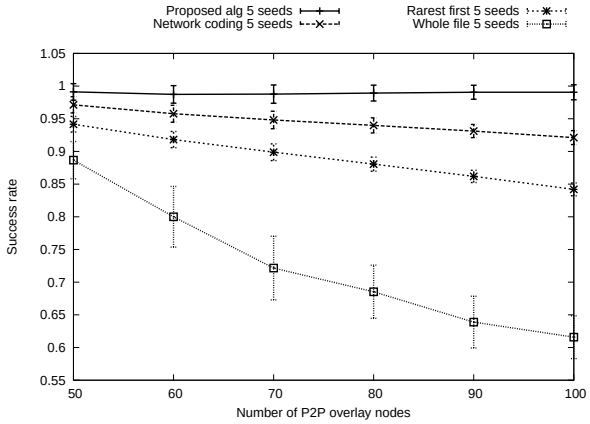
must obtain the file from the initial seed nodes. As the overlay size increases, the excessive traffic causes congestion around the initial seed nodes, and the success rate falls.

Figure 6.4 shows the fraction of peers that were successful in obtaining the 1 GB file. Due to the large file size, the whole–file system performed extremely poorly with not a single non–initial seed peer able to obtain the entire file. Since it takes a longer time for peers to download the file due to its larger size, more peers must try to download from the initial seeds. This causes severe congestion and eventually the seed nodes are forced to leave the overlay when their remaining energy falls below 10% of their capacity. The rarest first technique also performed more poorly when downloading a 1 GB file than a 100 MB file because there are more rare blocks to obtain. Initially, only the seed peers have the rarest blocks, so they are inundated with requests, causing congestion around those peers. This reduces the success rate for peers downloading the file. The network coding scheme and proposed algorithm both performed very well, identical to the 100 MB case. Because all file blocks are considered equivalent, it is possible for even some of the earliest downloading peers to obtain blocks from non–initial seed peers. Therefore, the success rate continues to remain at a high level.

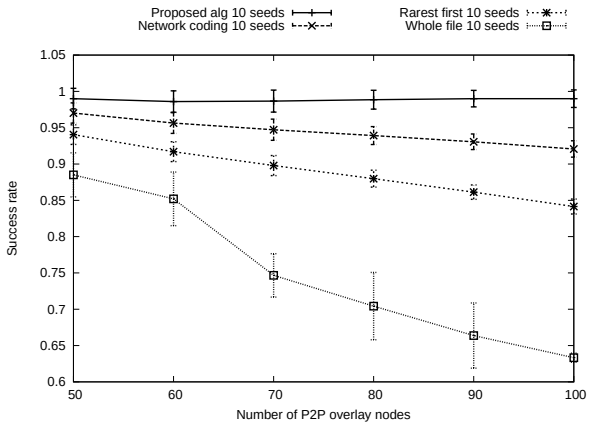
Figures 6.5 and 6.6 show the average time required to download the 100 MB and 1 GB files respectively. Only successful downloads are counted in the figures. The proposed algorithm performs the best, followed by the network coding only algorithm, and the rarest first system. Since no peers are able to successfully download the 1 GB content using the whole–file system, its data are not shown. The file download times do not change much as the overlay size increases because though the traffic



(a) 1 initial seed

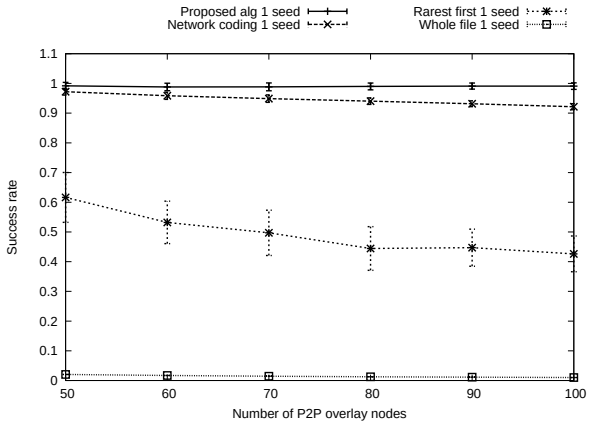


(b) 5 initial seeds

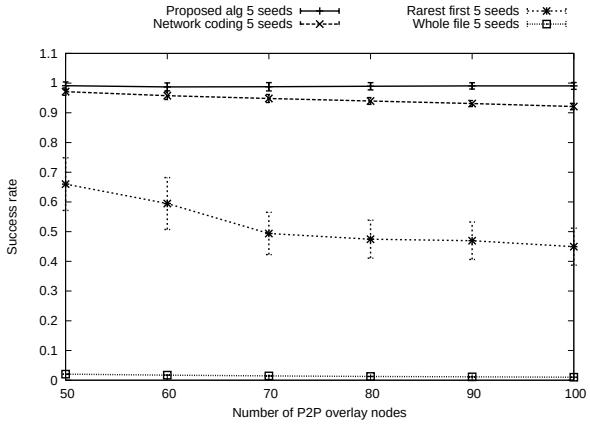


(c) 10 initial seeds

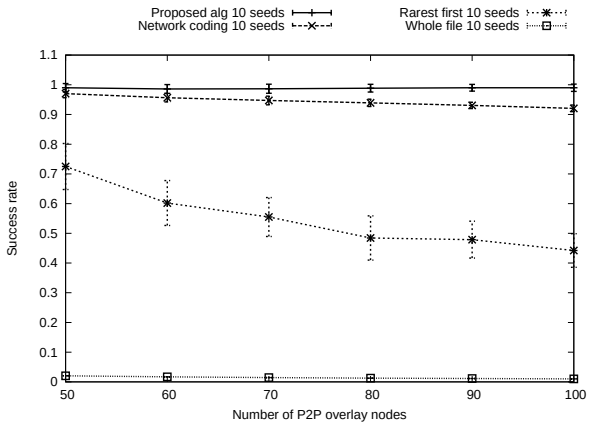
Figure 6.3: File download completion rate for 100 MB file



(a) 1 initial seed



(b) 5 initial seeds



(c) 10 initial seeds

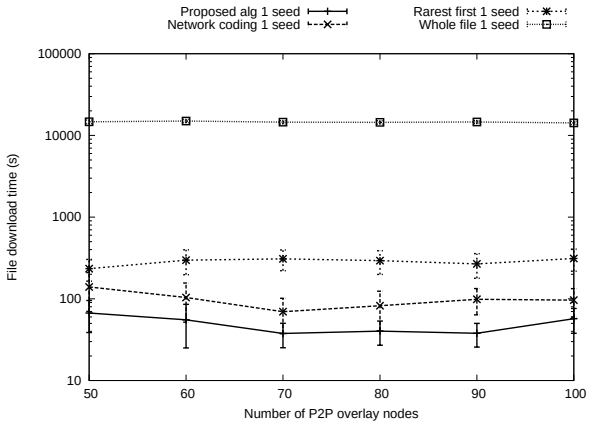
Figure 6.4: File download completion rate for 1 GB file

increases due to more peers trying to download, the peers able to supply file blocks also increases. As more initial seeds are available, the download time also falls since the availability of the file has increased within the network and more peers are able to supply blocks.

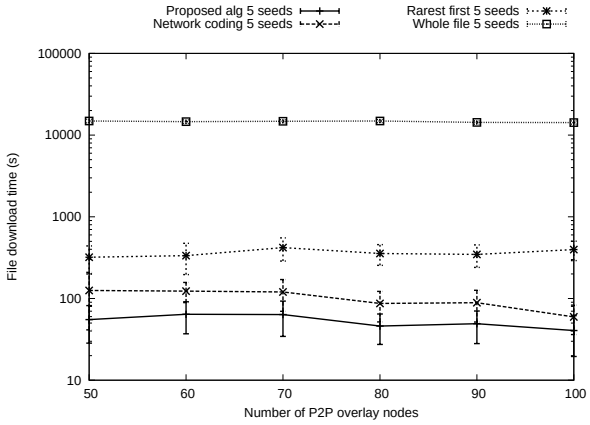
Figures 6.7 and 6.8 show the energy used by the initial seed nodes for the 100 MB and 1 GB file sizes respectively. The proposed algorithm has the lowest energy consumption because of its use of multicasting enables multiple clients to obtain blocks with a single data transfer. All multicasting overhead is included in the data. The network coding algorithm also performs well because the equivalency of blocks means that peers that were not initially seeds are able to contribute more quickly. The rarest first and whole–file systems perform the most poorly because they require the seed nodes to contribute many more blocks. As the overlay size increases, all systems show greater energy use because more peers are attempting to download the content. As more seeds are available, the average seeds' energy consumption falls because there are more alternatives for clients to choose from.

For the 1 GB file size, the whole–file system nearly always consumes the maximum possible energy. Since the whole file must be obtained from one seed, this node quickly runs out of energy, before it is able to successfully upload the full file to even a single non–seed peer. Even with a greater number of initial seeds the network congestion is too high, preventing peers from successfully obtaining the file and usually causing the seed to run out of energy. The rarest first technique also runs out of energy for a single seed when the overlay size increases. As more seeds are available, all the systems, other than whole–file, are able to reduce energy consumption, as expected.

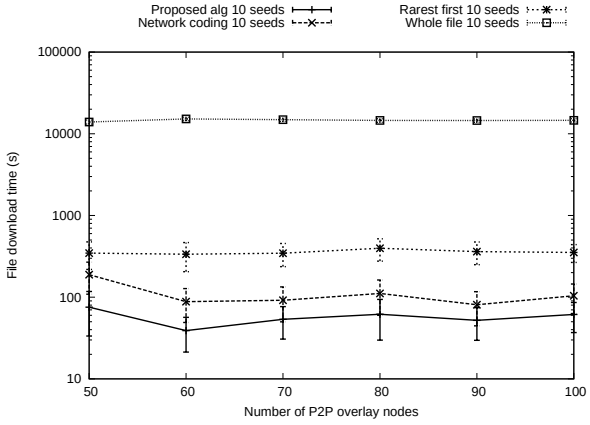
Figures 6.9 and 6.10 show how many blocks were uploaded, on average, by peers



(a) 1 initial seed

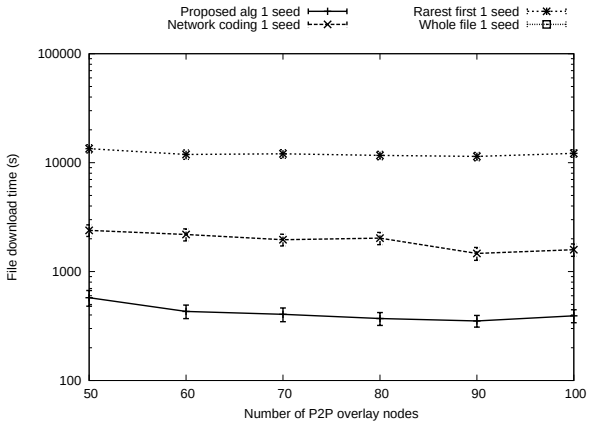


(b) 5 initial seeds

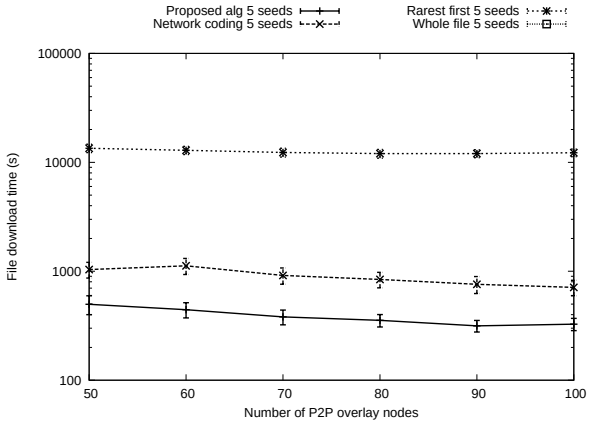


(c) 10 initial seeds

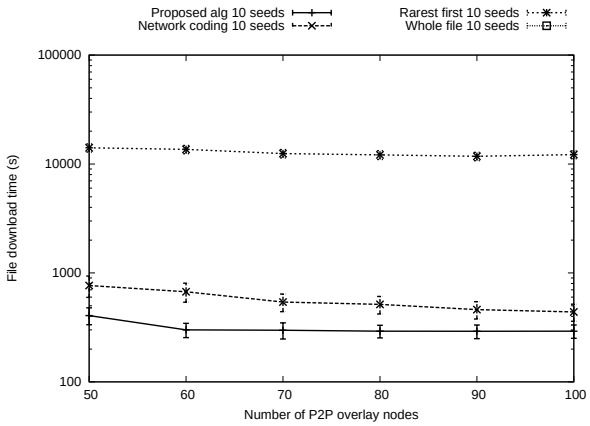
Figure 6.5: File download time in seconds for 100 MB file



(a) 1 initial seed

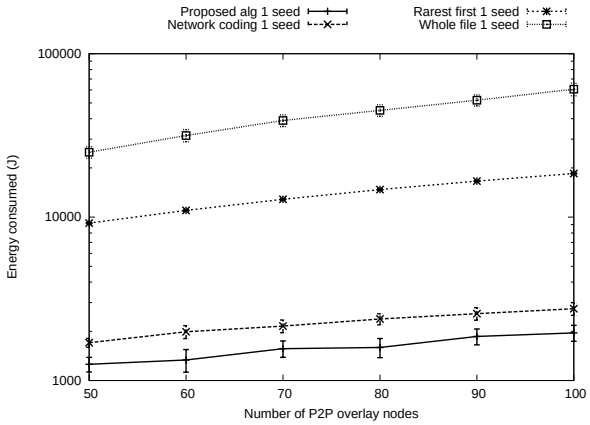


(b) 5 initial seeds

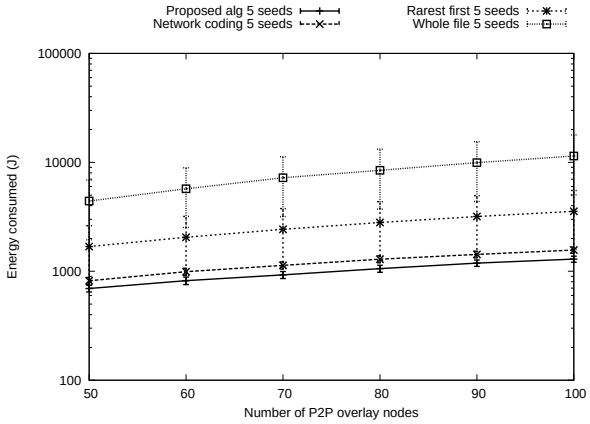


(c) 10 initial seeds

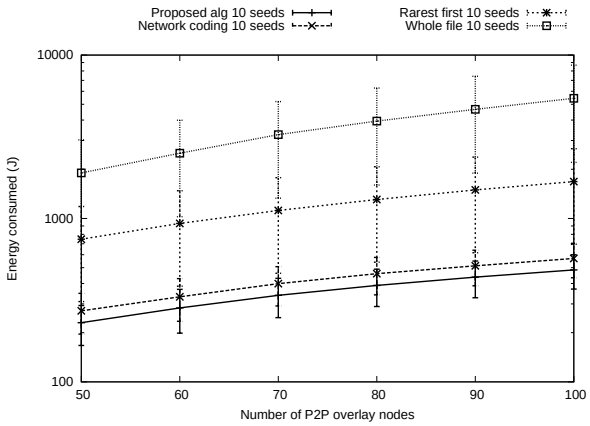
Figure 6.6: File download time in seconds for 1 GB file



(a) 1 initial seed

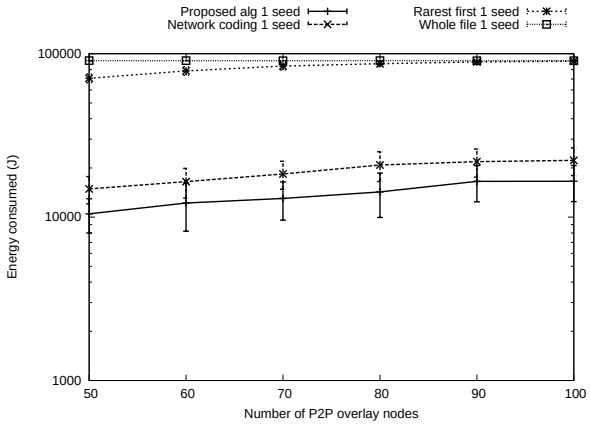


(b) 5 initial seeds

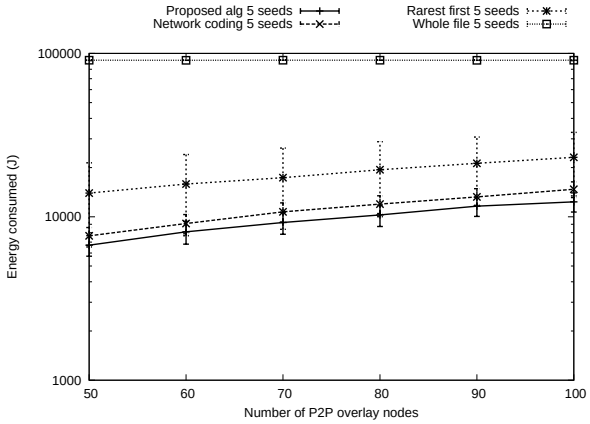


(c) 10 initial seeds

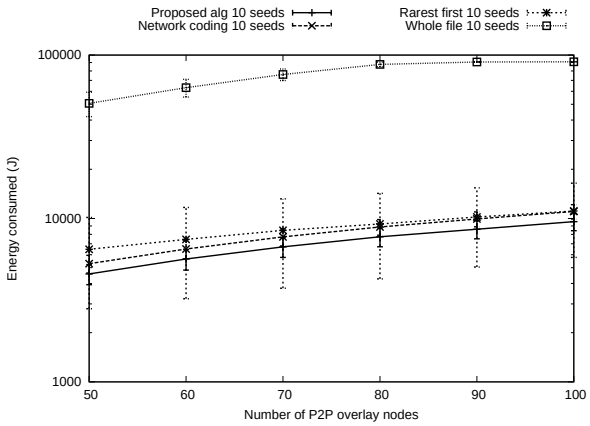
Figure 6.7: Energy consumed by initial seeds in Joules for 100 MB file



(a) 1 initial seed



(b) 5 initial seeds



(c) 10 initial seeds

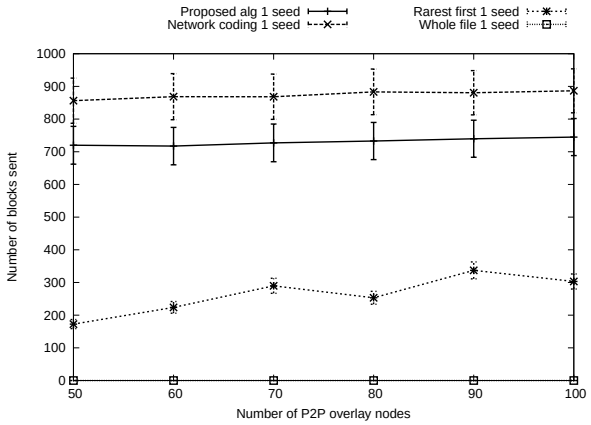
Figure 6.8: Energy consumed by initial seeds in Joules for 1 GB file

that are not initially seeds for the 100 MB and 1 GB file sizes respectively. In the whole–file system non–initial seed peers did not upload any blocks, meaning that all blocks were uploaded by the initial seeds. The proposed algorithm's non–initial seed nodes uploaded fewer blocks than the network coding algorithm due to the use of multicasting. This enabled a single block sent to by received by multiple receivers, reducing the number of sent blocks required. As the number of initial seeds increased, the average peer uploaded fewer blocks as expected.

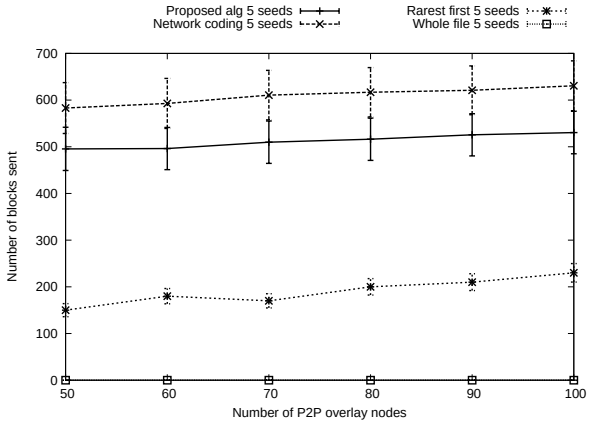
With the larger file size, the rarest first scheme caused non–initial seed peers to upload many more blocks than either of the two network coding schemes. Because a server might have the rarest block, it must upload that block many times before it is better spread through the overlay and peers get this updated information. This accounts for the larger confidence intervals seen, since a smaller number of peers upload far more blocks than the other peers. For the case of 10 initial seed nodes, the seeds uploaded all blocks for smaller network sizes.

Figures 6.11 and 6.12 shows the number of blocks sent by initial seed peers. As expected, the whole–file system required the seeds to upload the most, while the proposed algorithm required them to upload the least. The use of multicasting allows the proposed algorithm's seeds to send fewer blocks than using network coding alone. The very large confidence intervals for the whole–file and rarest first schemes show that the initial seeds upload highly varying numbers of blocks. Because the blocks are not considered equivalent, some seeds may have rarer blocks than others after a time, for the rarest block scheme. For the whole–file scheme, some seed peers are more centrally located in the network, so those seeds must upload more blocks.

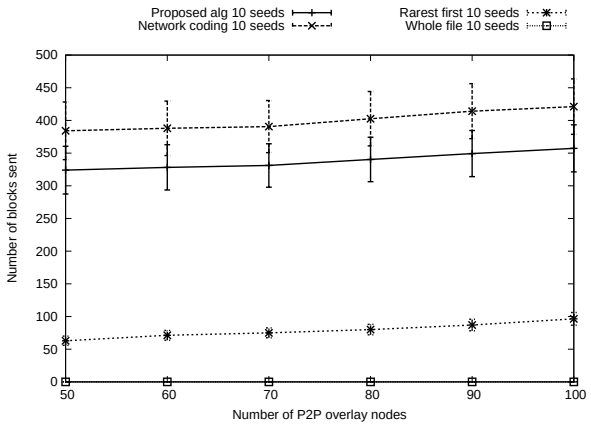
The maximum number of blocks that a peer can send peaks at just shy of half a



(a) 1 initial seed

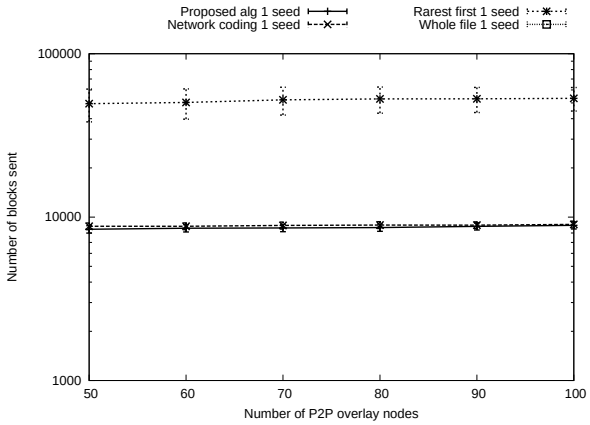


(b) 5 initial seeds

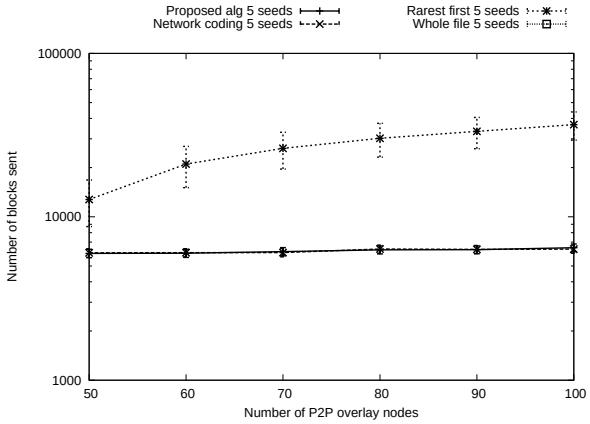


(c) 10 initial seeds

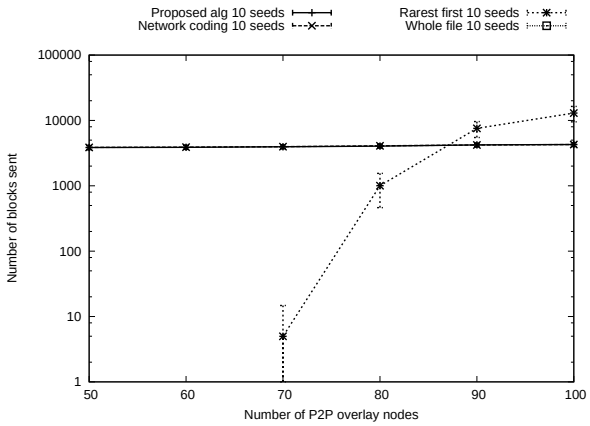
Figure 6.9: Blocks sent by non–initial seed peers for 100 MB file



(a) 1 initial seed



(b) 5 initial seeds



(c) 10 initial seeds

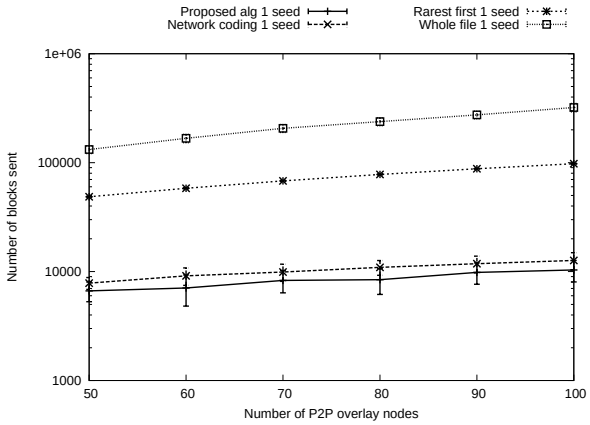
Figure 6.10: Blocks sent by non–initial seed peers for 1 GB file

million. At that point, it has consumed more than 90% of its energy and leaves the overlay. In the case of the 1 GB file, this level is attained for the rarest first system when a single initial seed is available, and for the whole–file system in all cases. This indicates that the burden of sharing the content is not well distributed through the overlay.

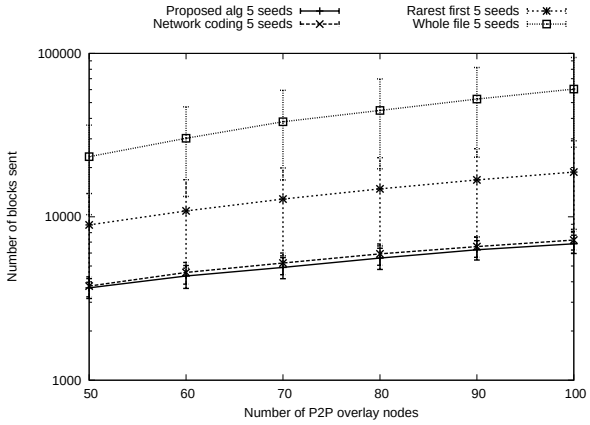
Figures 6.13 and 6.14 show Jain's fairness index values for the 100 MB and 1 GB file sizes respectively. For the case of 100 MB, the rarest first and whole–file systems exhibit poor fairness, confirming earlier results. For these systems, the initial seed peers upload a much larger share of the total than for the two network coding schemes. The network coding schemes show that the fairness decreases as the overlay size increases. This happens because clients attempt to download as many blocks as possible from a given server. Therefore, those peers which have more blocks will tend to upload more, even though all blocks are equivalent. This is desirable for clients because it means they need to download from fewer servers, and it also increases the chance of obtaining innovative blocks.

For the 1 GB case, the rarest first technique has much greater fairness than the 100 MB case, and it does not change much as the overlay size increases. This happens because the larger file size means that as the simulation progresses, there are many more blocks that are better spread throughout the overlay, so that clients spread their download requests more evenly.

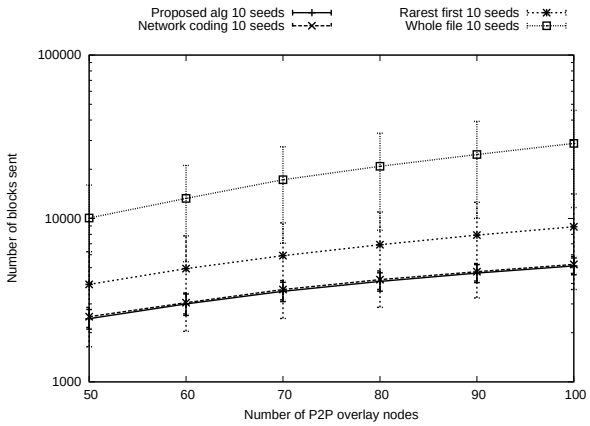
Figure 6.15 shows the number of initial seeds that have used all their energy and left the overlay by the time the simulation ends. The figure shows only the 1 GB case because no peers used all their energy with the 100 MB file. The whole–file system caused each seed to consume all its energy when there were either 1 or 5



(a) 1 initial seed

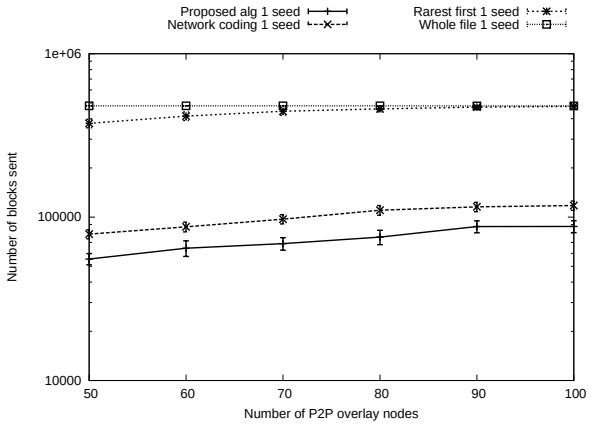


(b) 5 initial seeds

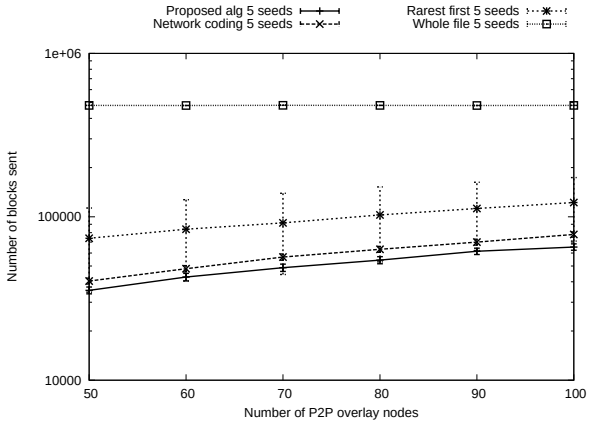


(c) 10 initial seeds

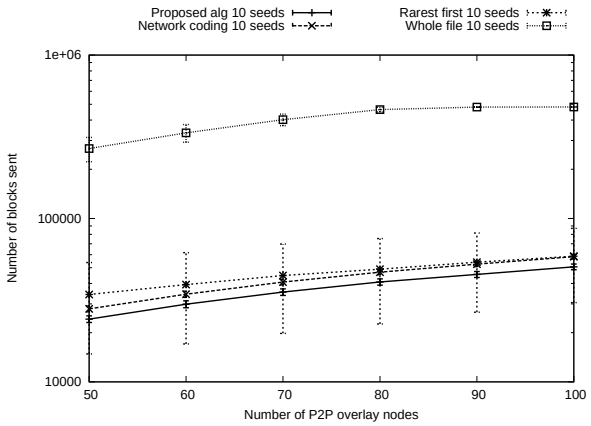
Figure 6.11: Blocks sent by initial seeds for 100 MB file



(a) 1 initial seed

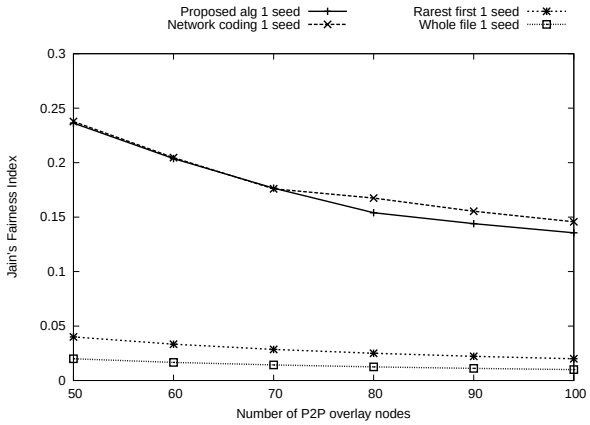


(b) 5 initial seeds

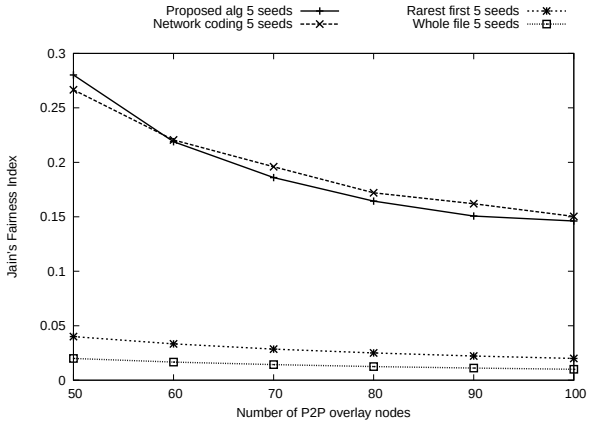


(c) 10 initial seeds

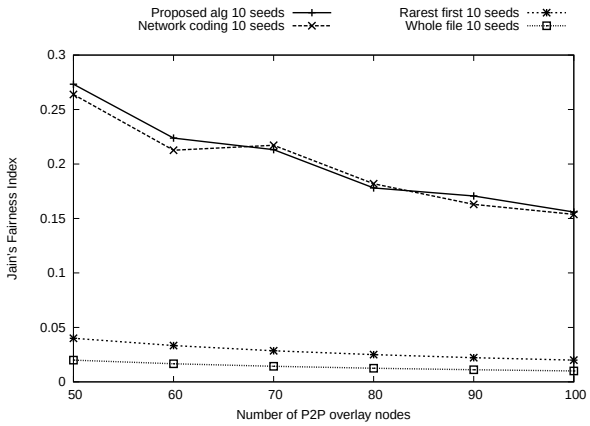
Figure 6.12: Blocks sent by initial seeds for 1 GB file



(a) 1 initial seed

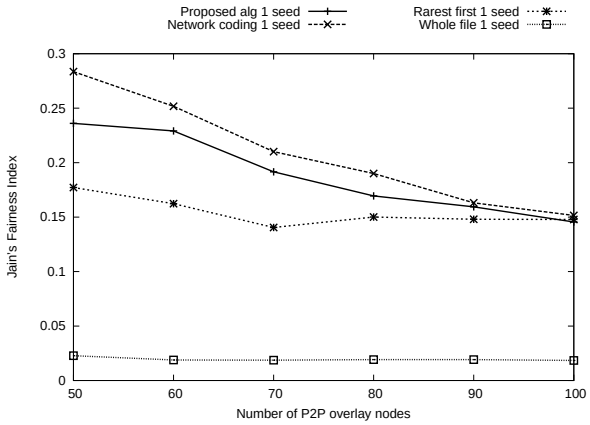


(b) 5 initial seeds

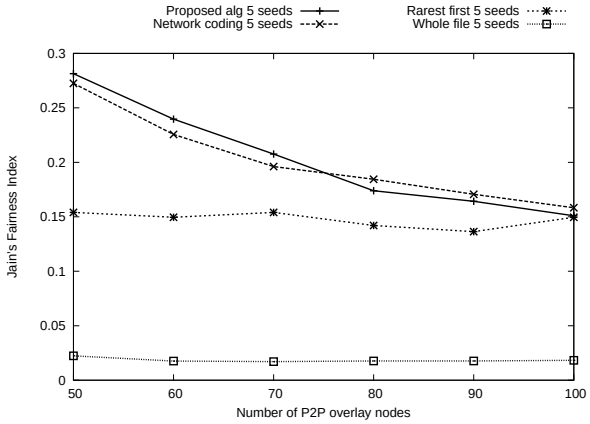


(c) 10 initial seeds

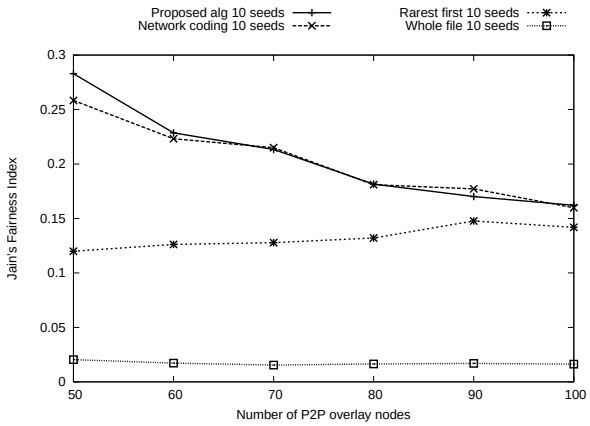
Figure 6.13: Jain's Fairness Index for 100 MB file



(a) 1 initial seed



(b) 5 initial seeds



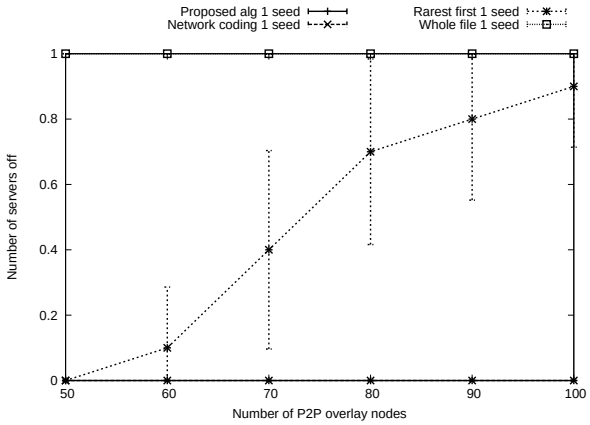
(c) 10 initial seeds

Figure 6.14: Jain's Fairness Index for 1 GB file

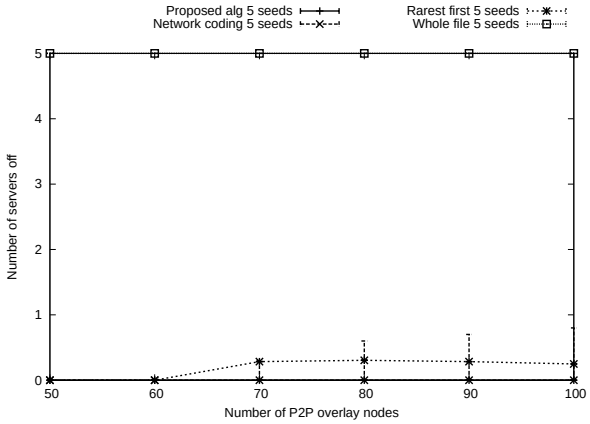
initial seeds. With 10 initial seeds the whole–file system still exhausted the energy of between 5 and 10 peers. Because these seeds used all their energy before they had successfully uploaded even a single copy of the content, no other peer was able to obtain the content. The rarest first scheme also had initial seeds using all their energy, though in all cases the number was low enough that other peers were able to obtain the complete content. Neither of the network coding–based schemes had initial seeds exceed their energy capacity.

6.5 Summary

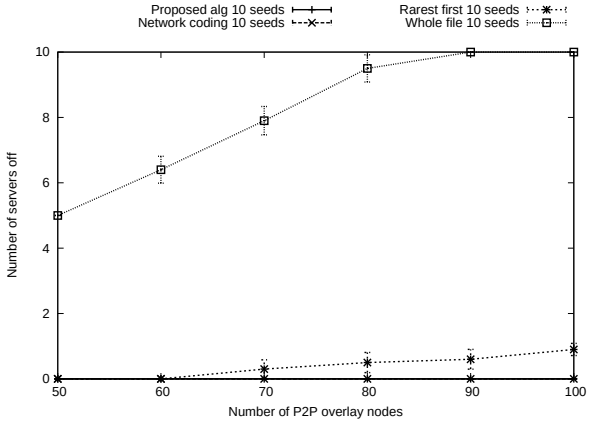
This chapter proposed an efficient content distribution scheme for P2P–MANETs that makes use of network coding and multicasting to provide a multipoint–to–multipoint distribution mechanism. The use of network coding brings about equivalency of all blocks, removing the rarest–block problem and decreasing download times. The use of multicasting reduces the number of transmissions required of seed peers, thus reducing their energy consumption. Simulation results show that the proposed scheme has a higher success rate, uses less energy, and has faster download times than the competing systems. The next chapter draws conclusions from this research and provides ideas for future research directions.



(a) 1 initial seed



(b) 5 initial seeds



(c) 10 initial seeds

Figure 6.15: Seeds using all their energy for 1 GB file

Chapter 7

Conclusions and Future Work

Peer–to–peer (P2P) networks are immensely popular among users and may be used for many different applications, such as file sharing, Voice–over–IP, gaming and instant messaging. Combining mobile ad hoc networks (MANETs) and peer–to–peer networks together so that a P2P overlay runs on a cooperative MANET is a natural evolution, since they share many similarities. Both are fully decentralized, must dynamically organize, must deal with frequent topology changes, must be resilient to failure, and must perform the routing function.

We believe that in the future, more and more users and will want to continue using P2P applications as they migrate to wireless devices and begin to use mobile, wireless, ad hoc networks as they become more prevalent as part of mesh and 4G networks. It is important, therefore, to develop a framework for P2P–MANETs because though P2P networks and MANETs share many similarities, simply adopting existing P2P overlay techniques and using them in MANETs is undesirable, since there also significant differences. Internet P2P networks support millions of simultaneous immobile users and function as overlays on the edge of the Internet. MANETs have fewer nodes

that are geographically close to one another and may be using resource–constrained devices, particularly in terms of energy, bandwidth, and delay.

This thesis studied the problem of designing a practical and effective P2P–MANET framework, and made several contributions to the research in this area. In this chapter, we summarize and discuss the conclusions from this thesis and provide directions for future work.

7.1 Summary of Contributions

We showed that existing research on peer–to–peer overlays has largely ignored the special considerations that mobile ad hoc networks require. We then proposed a framework for peer–to-peer overlays running on mobile ad hoc networks, consisting of several components that allow P2P–MANETs to be implemented in practice. The framework was designed to achieve and balance the following objectives:

- Support completely decentralized bootstrapping with high success rates
- Provide a computationally feasible topology control algorithm that considers neighbour energy levels and the stretch to all other peers
- Require users to contribute to the P2P overlay in return for using its services
- Select servers and paths to obtain services from in order to minimize the cost or download time
- Provide large file distribution with lower download times and less energy consumption than current techniques

In Chapter 3, we proposed a completely distributed bootstrapping mechanism for P2P–MANETs, the first of its kind. The proposed scheme used multicasting to allow nodes to query the MANET in order to determine which overlays were available to join. Prospective overlay neighbours then responded to the multicast group, and nodes cached the entries. This provided the original requesting node with a list of peers from which to choose to connect to.

The performance of the bootstrapping mechanism was tested under various overlay sizes with the results compared to flooding and several random address probing schemes. The results showed that the proposed scheme had the highest success rate, and received fast responses from other peers with reasonable overhead. The bootstrapping mechanism was implemented in Java, using the Microsoft Mesh Toolkit, and it was found to work well (see Appendix B).

In Chapter 4, we introduced a P2P–MANET creation game, based on the network creation game by Fabrikant *et al.* This game allowed the selfish peers to tradeoff the cost of maintaining links to neighbours against the stretch of the overlay. Using this game, it is possible to obtain the optimum minimum cost topology, as well as the Nash equilibrium, or stable topology, one in which no peer has an incentive to deviate. Solving the P2P–MANET creation game is NP–hard, so a heuristic algorithm was proposed. The heuristic used a similar cost function, but imposed a maximum degree constraint on each peer. The heuristic also allowed the overlay network to be created in steps, unlike the creation game, which had to be created all at once. This allowed the heuristic to better handle the inevitable topology changes of the P2P–MANET.

The performance of the proposed overlay topology control heuristic was determined under several overlay sizes and values of the single parameter α. It was shown

that as the cost of maintaining links increases, the heuristic algorithm reduces its neighbour count, while still trying to keep its total cost low. Approximations to the optimal minimum cost and Nash equilibrium were obtained via a random local search algorithm. The Nash equilibrium was often not able to be found and the heuristic algorithm compared quite favourably to the minimum cost, despite the fact that peers acted in a completely distributed manner and only sought to reduce their own costs.

In Chapter 5, we proposed a credit–based incentive scheme to encourage peers to cooperate in the P2P–MANET. The scheme allowed peers to be compensated for sharing data as well as forwarding traffic in the network. The compensation was in the form of virtual credits, which can then be used to obtain services, such as files, from the overlay. The scheme was dynamically priced, allowing peers to select their own prices, based on the demand for the file as well as traffic demands on the peer. The scheme was compared to fixed price and unpriced schemes, and it was shown to permit a large number of downloads, while maintaining significantly greater fairness than the competing schemes. The incentive scheme was implemented in Java, using the Microsoft Mesh Toolkit, and in conjunction with the implementation of the bootstrapping mechanism, was shown to successfully price files as per the user's preferences (see Appendix B).

In addition, we proposed a mixed integer linear program that selected the optimal paths for a download, given a set of possible paths. The program allowed peers to download blocks from multiple servers simultaneously, and even use multiple paths to the same server. The download time could be minimized subject to a maximum budget, or the download cost minimized subject to a maximum time. Because the optimal solution requires complete *a priori* knowledge of all file requests, paths, and

block counts at servers, it is impractical for use in a real P2P–MANET. Therefore, we also proposed a greedy heuristic algorithm that allows the user to tradeoff between the cost and download time of the file. The heuristic, like the linear program, provides a list of how many blocks to download from which paths, to satisfy the requirements of the user. The performance of the heuristic was very favourable when compared to the optimal solution, in addition to its flexibility and practicality.

In Chapter 6, we introduced an efficient content distribution scheme for P2P–MANETs. The scheme used network coding in addition to multicasting to transfer blocks of data between peers. Network coding created an equivalence of all file blocks, allowing nodes to obtain any blocks they could find from servers, without concern for locating specific blocks. This technique eliminated the rarest block problem. Furthermore, multicasting the blocks allowed servers to efficiently deliver them to many receivers and reduced transmissions at the server node.

The performance of the proposed content distribution scheme was compared to downloading the entire file from a single seed, downloading blocks from multiple servers, and network coding without multicasting. It was shown that the proposed scheme consumed less energy, provided speedier downloads, and had a greater success rate than the competing algorithms.

7.2 Future Research Directions

There are several ways in which the research in this thesis can be furthered. We now examine some possibilities.

In Chapter 3 we considered bootstrapping in the context of completely infrastructureless mobile ad hoc networks. The emergence of mixed infrastructure and

infrastructureless networks, such as wireless mesh and 4G networks, allows the bootstrapping mechanism to be enhanced to support these new types of networks. Since an infrastructure component exists, it may be possible to combine the existing research on boostrapping and combine it with the technique described in this thesis.

A downside of the proposed algorithm is that its performance and message complexity relies largely on the underlying multicast routing protocol. A possible research direction would be to modify the bootstrap algorithm to allow those parts of the network with a low density of overlay nodes to not participate in the multicast algorithm and instead allow peers to search and discover others through the use of advertisements. This technique would likely result in slower discovery, but would also reduce reduce energy consumption and reduce message overhead. Another option would be to allow a group of nodes to act as a distributed registry. Overlay members would register their status with the registry, and newly joining nodes could contact the registry to obtain a list of potential neighbours.

In Chapter 4 we examined a topology control heuristic that balanced the cost of maintaining links to neighbours with the cost of the stretch to all other peers. Our link cost was based on the energy of the potential neighbour. Another possibility is to account for network traffic instead of neighbour energy. If a large amount of traffic is transiting a particular link, then it may make more sense to "lay down" the link instead of routing it through a possibly less efficient path. The downside to this approach is that the link traffic must be measured and communicated between peers, which may itself cause excessive traffic.

Another possibility would be to develop an overlay algorithm that is a hybrid of the structured and unstructured types, i.e., not as rigid as a traditional structured

overlays nor as loose as an unstructured overlay. This overlay must support peer mobility through more optimistic timeout and failure strategies, and should also consider energy consumption. The benefit of such an overlay would be the determination of an upper bound on lookup and routing time, as well as a reduction in message overhead due to the flooding of queries. Combining the MANET with infrastructured networks makes this direction more feasible.

A topology control algorithm that focuses on maximizing utilization of the substrate would be beneficial from several points of view, including energy consumption and delay. Such a topology might better support streaming media and multicasting.

In Chapter 5 we presented an incentive scheme to promote sharing and compensate peers for their contributions to the overlay. One problem that may arise is that some peers will accrue large amounts of credit, but not use it. This may "starve" other peers and prevent them from legitimate use of the overlay. These peers may be willing to contribute to earn credits, but no opportunity arises for them to do so. One possible solution is to allow nodes to borrow credits from others, with a future repayment including interest. Peers may offer different interest rates depending on the perceived credit–worthiness of the borrower, and a borrower may "shop" around for the best deal. Another possible solution that may be combined with the first, is that the value of each credit decreases with time, as with price inflation. Therefore nodes lose out by hoarding and are encouraged to spend their credits before the value drops too far.

To prevent nodes from cheating one another, a trust system may be overlaid on the credit–based system. This trust system may take the form of one of the schemes discussed in Chapter 2, or it may be a rotating group of peers that maintain trust

information that is combined to determine a node's trustworthiness. A means of enforcing fairness and accountability, requiring all servers to provide a fair proportion of the service would be useful. Determining what constitutes fairness and what to do in the event that a server cannot perform the service due to low energy or mobility is a challenging problem.

Also presented in Chapter 5 was a path selection algorithm. The algorithm allowed peers to choose which path was best for their preferences when downloading a file. In the future, this algorithm can be extended to support streaming and real–time data. These data are time–sensitive and therefore we cannot consider only the price and download time. It is necessary to also ensure that the data arrive in the correct order and in a timely fashion. This is a particularly challenging problem since it must take into account the uplink and downlink bandwidth and delay of all MANET nodes on the path from the source to the destination.

In Chapter 6 we discussed a content–distribution scheme that uses network coding and multicasting. A future direction for this research is to add support for streaming data. There are many possible ways to pursue this goal, including the use of a push-based tree, a push-pull hybrid scheme, the use of cross–links in the multicast tree, or a gossip-based system to allow non-seed nodes to obtain blocks.

Another possible direction of research is to add support for Quality of Service mechanisms (QoS). Using a bandwidth allocation scheme together with scheduling algorithms, it may be possible to even further improve the performance of the content distribution scheme. The cost function can be modified to accomodate higher service levels, which will cost more, or allow peers to download in bulk at a lower unit price.

We examined the performance of all components of the proposed P2P–MANET

framework on 100–node MANETs with various overlay sizes, but we expect that they will scale well as the MANET and overlay sizes increase, as indicated by the outcome of our simulation experiments. We also expect that the results will be similar for more sparsely populated networks. In the future, this could be verified with further simulation on sparser and also larger networks.

Finally, the entire P2P–MANET system, in whole and in part, may be modelled mathematically and analyzed in terms of performance and availability. A security mechanism could be added to each of the framework components in the case of non–cooperative MANETs.

Bibliography

[1] Eytan Adar and Bernardo A. Huberman. Free riding on Gnutella. *First Monday*, 5(10), 2000.

[2] Micah Adler, Rakesh Kumar, Keith Ross, Dan Rubenstein, David Turner, and David D. Yao. Optimal peer selection in a free-market peer-resource economy. In *Second Workshop on Economics of Peer-to-Peer Systems*, June 2004.

[3] Micah Adler, Rakesh Kumar, Keith Ross, Dan Rubenstein, David Turner, and David D. Yao. Optimal peer selection for p2p downloading and streaming. In *IEEE INFOCOM*, pages 1538–1549, 2005.

[4] Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, and Raymond W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.

[5] Kostas G. Anagnostakis and Michael B. Greenwald. Exchange-based incentive mechanisms for peer-to-peer file sharing. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 524–533, 2004.

[6] Luzi Anderegg and Stephan Eidenbenz. Ad hoc-vcg: A truthful and cost-efficient routing protocol for mobile ad hoc networks with selfish agents. In *ACM MobiCom'03*, pages 245–259, San Diego, CA, 2003.

[7] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in P2P systems. *Communications of the ACM*, 46(2):43–48, 2003.

[8] Amit Banerjee and Chung-Ta King. Building ring-like overlays on wireless ad hoc and sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 20(11):1553–1566, November 2009.

[9] Sorav Bansal and Mary Baker. Observation-based cooperation enforcement in ad hoc networks. Technical report, Stanford University, July 2003.

[10] Stefano Basagni, Marco Conti, Silvia Giordano, and Ivan Stojmenovic, editors. *Mobile Ad Hoc Networking*. IEEE Press, 2004.

[11] BitTorrent. http://www.bittorrent.com/.

[12] Azzedine Boukerche, Anis Zarrad, and Regina Araujo. A cross-layer approach-based gnutella for collaborative virtual environments over mobile ad-hoc networks. *IEEE Transactions on Parallel and Distributed Systems, pre–print*, 2009.

[13] Sonja Buchegger and Jean-Yves Le Boudec. Performance analysis of the confidant protocol: Cooperation of nodes – fairness in dynamic ad-hoc networks. In *IEEE/ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC)*, pages 226–236, June 2002.

[14] Sonja Buchegger and Jean-Yves Le Boudec. A robust reputation system for p2p and mobile ad-hoc networks. In *Second Workshop on the Economics of Peer–to–Peer Systems*, 2004.

[15] Levente Buttyán and Jean-Pierre Hubaux. Stimulating cooperation in self-organizing mobile ad hoc networks. *Mobile Networks and Applications*, 8(5):579–592, 2003.

[16] John Byers, Jeffrey Considine, Michael Mitzenmacher, and Stanislav Rost. Informed content delivery across adaptive overlay networks. *IEEE/ACM Transactions on Networking (TON)*, 12:767–780, October 2004.

[17] John Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege. A digital fountain approach to reliable distribution of bulk data. *ACM SIGCOMM Computer Communication Review*, 28:56–67, October 1998.

[18] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. One ring to rule them all: service discovery and binding in structured peer-to-peer overlay networks. In *ACM SIGOPS European workshop*, pages 140–145, 2002.

[19] Philip A. Chou and Yunnan Wu. Network coding for the internet and wireless networks. *IEEE Signal Processing Magazine*, pages 77–85, September 2007.

[20] Philip A. Chou, Yunnan Wu, and Kamal Jain. Practical network coding. In *Allerton Conference Communication, Control and Computing*, October 2003.

[21] Byung-Gon Chun, Rodrigo Fonseca, Ion Stoica, and John Kubiatowicz. Characterizing selfishly constructed overlay routing networks. In *IEEE INFOCOM*, volume 2, pages 1329–1339, 2004.

[22] Michael Conrad and Hans-Joachim Hof. A generic, self-organizing, and distributed bootstrap service for peer-to-peer networks. In *IWSOS 2007, LNCS 4725*, pages 59–72, 2007.

[23] Marco Conti, Enrico Gregori, and Gaia Maselli. Towards reliable forwarding for ad hoc networks. In *Personal Wireless Communications (PWC)*, volume 2775 of *Lecture Notes in Computer Science*, pages 790–804, 2003.

[24] Curt Cramer and Thomas Fuhrmann. Bootstrapping chord in ad hoc networks: Not going anywhere for a while. In *IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW)*, pages 168–172, 2006.

[25] Curt Cramer and Thomas Fuhrmann. Performance evaluation of chord in mobile ad hoc networks. In *MobiShare*, September 2006.

[26] Diego N. da Hora, Daniel F. Macedo, Leonardo B. Oliveira, Isabela G. Siqueira, Antonio A.F. Loureiro, José M. Nogueira, and Guy Pujolle. Enhancing peer-to-peer content discovery techniques over mobile ad hoc networks. *Computer Communications*, 32:1445–1459, August 2009.

[27] Franca Delmastro. From Pastry to CrossROAD: CROSS-layer Ring Overlay for AD hoc networks. In *Pervasive Computing and Communications Workshops (PerCom)*, pages 60–64, 2005.

[28] Zoran Despotovic and Karl Aberer. Maximum likelihood estimation of peers' performance in p2p networks. In *Second Workshop on the Economics of Peer–to–Peer Systems*, June 2004.

[29] Boubacar Kimba dit Adamou, Shihong Zou, Yihui Ma, Shiduan Cheng, and Dandan Hao. Effective file sharing mechanism with network coding in wireless mesh networks. In *Wireless Communications, Networking and Mobile Computing*, pages 1677–1680, September 2007.

[30] Debojyoti Dutta, Ashish Goel, Ramesh Govindan, and Hui Zhang. The design of a distributed rating scheme for peer-to-peer systems. In *First Workshop on the Economics of Peer–to–Peer Systems*, June 2003.

[31] Alex Fabrikant, Ankur Luthra, Elitza Maneva, Christos H. Papadimitriou, and Scott Shenker. On a network creation game. In *ACM Symposium on Principles of Distributed Computing*, pages 347–351, 2003.

[32] Laura Marie Feeney. An energy–consumption model for performance analysis of routing protocols for mobile ad hoc networks. *Mobile Networks and Applications*, 6(3):239–250, 2001.

[33] Christina Fragouli, Jean-Yves Le Boudec, and Jorg Widmer. Network coding: An instant primer. *Computer Communication Review*, 36(1):63–68, January 2006.

[34] Christos Gkantsidis, John Miller, and Pablo Rodriguez. Anatomy of a p2p content distribution system with network coding. In *International workshop on Peer-To-Peer Systems (IPTPS)*, February 2006.

[35] Christos Gkantsidis, John Miller, and Pablo Rodriguez. Comprehensive view of a live network coding p2p system. In *ACM SIGCOMM conference on Internet measurement*, pages 177–188, October 2006.

[36] Christos Gkantsidis and Pablo Rodriguez Rodriguez. Network coding for large scale content distribution. In *IEEE Infocom*, pages 2235–2245, March 2005.

[37] Gnutella. The annotated gnutella protocol specification v0.4. http://rfc–gnutella.sourceforge.net/developer/stable/index.html.

[38] Dominik Grolimund, Luzius Meisser, Stefan Schmid, and Roger Wattenhofer. Havelaar: A robust and efficient reputation system for active peer-to-peer systems. In *First Workshop on the Economics of Networked Systems (NetEcon)*, 2006.

[39] Ingo Gruber, Rüdiger Schollmeier, and Wolfgang Kellerer. Performance evaluation of the mobile peer-to-peer service. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 363–371, April 2004.

[40] Rohit Gupta and Arun K. Somani. An incentive driven lookup protocol for chord-based peer-to-peer (p2p) networks. In *International Conference on High Performance Computing (HiPC)*, volume 3296 of *Lecture notes in Computer Science*, pages 8–18, 2004.

[41] Anwar Al Hamra, Chadi Barakat, and Thierry Turletti. Network coding for wireless mesh networks: A case study. In *World of Wireless, Mobile and Multimedia Networks*, pages 103–114, 2006.

[42] Seung Chul Han and Ye Xia. Constructing an optimal server set in structured peer-to-peer networks. *IEEE Journal on Selected Areas in Communications*, 25(1):170–178, January 2007.

[43] Hossam Hassanein, Yu Yang, and Afzal Mawji. A new approach to service discovery in wireless mobile ad hoc networks. *Int. J. Sensor Networks*, 2(1/2):135–145, 2007.

[44] Y. Charlie Hu, Saumitra M. Das, and Himabindu Pucha. Exploiting the synergy between peer-to-peer and mobile ad hoc networks. In *USENIX Association 9th Workshop on Hot Topics in Operating Systems*, pages 37–42, 2003.

[45] Yang hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communications special issue on Network Support for Multicast Communications*, 20:1456–1471, October 2002.

[46] D. Johnson, Y. Hu, and D. Maltz. The dynamic source routing protocol (DSR) for mobile ad hoc networks for IPv4. RFC 4728, February 2007.

[47] Pradnya Karbhari, Mostafa Ammar, Amogh Dhamdhere, Himanshu Raj, George Riley, and Ellen Zegura. Bootstrapping in Gnutella: a measurement study. In *Passive and Active Measurement Workshop, LNCS 3015*, pages 22–32, 2004.

[48] Sachin Katti, Dina Katabi, Wenjun Hu, Hariharan Rahul, and Muriel Médard. The importance of being opportunistic: Practical network coding for wireless environments. In *Allerton Conference Communication, Control and Computing*, 2005.

[49] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. Xors in the air: Practical wireless network coding. *ACM SIGCOMM Computer Communication Review*, 36(4):243–254, October 2006.

[50] Alexander Klemm, Christoph Lindemann, and Oliver P. Waldhorst. A special-purpose peer-to-peer file sharing system for mobile ad hoc networks. In *IEEE Vehicular Technology Conference (VTC)*, volume 4, pages 2758–2763, October 2003.

[51] Mirko Knoll, Arno Wacker, Gregor Schiele, and Torben Weis. Decentralized bootstrapping in pervasive applications. In *IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW)*, pages 589–592, 2007.

[52] Pandeli Kolomitro. Service discovery system for use in wireless mesh networks. Queen's University, CISC 499 report, 2008.

[53] Dejan Kostić, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. *ACM SIGOPS Operating Systems Review*, 37:282–297, December 2003.

[54] H. T. Kung and Chun-Hsin Wu. Differentiated admission for peer-to-peer systems: Incentivizing peers to contribute their resources. In *First Workshop on the Economics of Peer–to–Peer Systems*, June 2003.

[55] Jim Kurose and Keith Ross. *Computer Networking: A Top–Down Approach, 5th ed.* Addison–Wesley, 2010.

[56] Paul Lanyon. An incentivized peer-to-peer file sharing application. Queen's University, CISC 499 report, 2008.

[57] Sung Ju Lee, William Su, and Mario Gerla. On–demand multicast routing protocol in multihop wireless mobile networks. *Mobile Networks and Applications*, 7(6):441–453, 2002.

[58] Shuo-Yen Robert Li, Raymond W. Yeung, and Ning Cai. Linear network coding. *IEEE Transactions on Information Theory*, 49(2):371–381, 2003.

[59] Wei-Cherng Liao, Fragkiskos Papadopoulos, and Konstantinos Psounis. An efficient algorithm for resource sharing in peer-to-peer networks. In *Networking Technologies, Services, and Protocols (Networking)*, volume 3976 of *Lecture Notes in Computer Science*, pages 592–605, 2006.

[60] Yunhao Liu. A two–hop solution to solving topology mismatch. *IEEE Transactions on Parallel and Distributed Systems*, 19(11):1591–1600, November 2008.

[61] Sergio Marti, T.J. Giuli, Kevin Lai, and Mary Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *International Conference on Mobile Computing and Networking (MobiCom)*, pages 255–265, Boston, MA, 2000.

[62] Benjamin David McBride. Constructing traffic-aware overlay topologies: A machine learning approach. Master's thesis, Kansas State University, 2007.

[63] Pietro Michiardi and Refik Molva. Core: A collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks. In *Sixth Joint Working Conference on Communications and Multimedia Security*, pages 107–121, 2002.

[64] Microsoft Research. Mesh networking academic resource toolkit. http://research.microsoft.com/en–us/projects/mesh/, 2007.

[65] Alvin Mok, Bina Mistry, Eric Chung, and Baochun Li. Fair: Fee arbitrated incentive architecture in wireless ad hoc networks. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 38–47, 2004.

[66] Tim Moreton and Andrew Twigg. Trading in trust, tokens, and stamps. In *First Workshop on the Economics of Peer–to–Peer Systems*, June 2003.

[67] Thomas Moscibroda, Stefan Schmid, and Roger Wattenhofer. On the topologies formed by selsh peers. In *5th International Workshop on Peer–to–Peer Systems*, February 2006.

[68] ns 2. The network simulator. http://www.isi.edu/nsnam/ns/.

[69] Thanasis G. Papaioannou and George D. Stamoulis. An incentives' mechanism promoting truthful feedback in peer-to-peer systems. In *IEEE International Symposium on Cluster Computing and the Grid*, volume 1, pages 275–283, 2005.

[70] S.-T. Park, A. Khrabrov, D.M. Pennock, S. Lawrence, C.L. Giles, and L.H. Ungar. Static and dynamic analysis of the internet's susceptibility to faults and attacks. In *IEEE INFOCOM*, volume 3, pages 2144–2154, 2003.

[71] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing. RFC 3561, July 2003.

[72] Charles E. Perkins and Elizabeth M. Royer. Multicast operation of the ad-hoc on-demand distance vector routing protocol. In *ACM/IEEE MobiCom*, pages 207–218, Seattle, WA, 1999.

[73] Dayou Qian, Chi Zhou, and Jinsong Zhang. Cooperation enforcement in ad hoc networks with penalty. In *IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS)*, 2005.

[74] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2001.

[75] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[76] Rüdiger Schollmeier, Ingo Gruber, and Florian Niethammer. Protocol for peer-to-peer networking in mobile environments. In *International Conference on Computer Communications and Networks (ICCCN)*, pages 121–127, October 2003.

[77] Tara Small, Baochun Li, and Ben Liang. Topology affects the efficiency of network coding in peer-to-peer networks. In *IEEE International Conference on Communications*, pages 5591–5597, May 2008.

[78] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. In *ACM SIGCOMM*, pages 149–160, September 2001.

[79] Yasuo Tamura, Shoji Kasahara, Yutaka Takahashi, Satoshi Kamei, and Ryoichi Kawahara. Inconsistency of logical and physical topologies for overlay networks

and its effect on file transfer delay. *Performance Evaluation*, 65(10):725–741, October 2008.

[80] Mohammed Tarique, Kemal E. Tepe, Sasan Adibi, and Shervin Erfani. Survey of multipath routing protocols for mobile ad hoc networks. *Journal of Network and Computer Applications*, 32:1125–1143, November 2009.

[81] Ian J. Taylor. *From P2P to Web Services and Grids: Peers in a Client/Server World*. Springer-Verlag, 2005.

[82] Hal R. Varian. *Intermediate Microeconomics, a Modern Approach, 6th ed.* W. W. Norton & Company, 2003.

[83] Vivek Vishnumurthy, Sangeeth Chandrakumar, and Emin Gün Sirer. Karma: A secure economic framework for peer-to-peer resource sharing. In *First Workshop on the Economics of Peer–to–Peer Systems*, June 2003.

[84] Wi–Fi Alliance. Wi–Fi Alliance announces groundbreaking specification to support direct Wi–Fi connections between devices. http://www.wi–fi.org/news_articles.php?f=media_news&news_id=909, 2009.

[85] Rolf Winter, Thomas Zahn, and Jochen Schiller. DynaMO: A topology-aware P2P overlay network for dynamic, mobile ad-hoc environments. *Telecommunication Systems*, 27(2–4):321–345, October 2004.

[86] Krit Wongrujira, Tim Hsin-ting, and Aruna Seneviratne. Incentive service model for p2p. In *ACS/IEEE International Conference on Computer Systems and Applications*, pages 81–87, 2005.

[87] Beverly Yang, Sepandar Kamvar, and Hector Garcia-Molina. Addressing the non-cooperation problem in competitive p2p systems. In *First Workshop on Economics of P2P Systems*, June 2003.

[88] Younghwan Yoo, Sanghyun Ahn, and Dharma P. Agrawal. A credit-payment scheme for packet forwarding fairness in mobile ad hoc networks. In *IEEE International Conference on Communications (ICC)*, volume 5, pages 3005–3009, 2005.

[89] Thomas Zahn and Jochen Schiller. MADPastry: A DHT substrate for practicably sized MANETs. In *IEEE Workshop on Applications and Services in Wireless Networks (ASWN)*, June 2005.

[90] Sheng Zhong, Jiang Chen, and Yang Richard Yang. Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks. In *IEEE INFOCOM*, volume 3, pages 1987–1997, San Francisco, March 2003.

[91] Danyu Zhu and Matt Mutka. Promoting cooperation among strangers to access internet services from an ad hoc network. In *IEEE Pervasive Computing and Communications (PERCOM)*, 2004.

Appendix A

MANET Routing

There are many MANET routing protocols in existence. In this appendix, we briefly discuss AODV [71], MAODV [72], and DSR [46], which are the ones referred to in this thesis.

A.1 AODV

Ad hoc On–Demand Distance Vector (AODV) routing is a reactive, distance vector, unicast routing protocol. Reactive protocols establish routes on–demand, as opposed to proactive routing protocols, which determine routes before they are needed. Distance vector protocols are distributed protocols that do not require global information, and in which nodes inform neighbours of topology changes. These changes are propagated throughout the network, neighbour–to–neighbour.

When a route to a destination is needed, the source broadcasts a Route Request (RREQ) message via UDP. When the RREQ reaches the destination node or an intermediate node that has a valid route, a Route Reply (RREP) message is unicasted

back to the source node. When a node receives a RREQ, it caches the route back to the source, so that in the future it may be able to send a RREP. When a node detects that a link has failed, it sends a Route Error (RERR) message to other nodes.

AODV uses destination sequence numbers for route entries in order to avoid routing loops. The destination sequence number is generated by the destination and is included with route information it sends to requesting nodes. By using the most recent known sequence number, nodes are able to choose more up–to–date routes.

A.2 MAODV

Multicast AODV (MAODV) is an extension of the AODV routing protocol that provides support for multicasting. Multicast routes are discovered on–demand, as in AODV. The route discovery process is the same as in AODV. Nodes send a RREQ to join a multicast group. The destination address in the RREQ packet is the multicast group address. Only multicast group members may respond to the RREQ with a RREP message. The set of routes determined by the multicast group nodes forms the multicast tree.

The first member of a multicast group becomes the group leader. The leader is responsible for maintaining the multicast group sequence number and periodically broadcasting a Group Hello message. When a node tries to join a multicast group it may receive multiple join points. The node selects one of these points and sends a Multicast Activation (MACT) message to it, indicating that this is the place in the tree where the new node has joined, and activating that route. This ensures that the multicast tree does not have multiple paths to any node. When a multicast group member receives a packet addressed to the group, it multicasts the packet to its next

hops. These are the routes that have been activated.

In addition to the Route Table required in AODV, nodes must also maintain a Multicast Route Table, which contains information about the multicast group address, group leader's IP address, and the group sequence number.

A.3 DSR

Dynamic Source Routing (DSR) is an on–demand routing protocol. It uses source routing, so the entire route is determined at the source node, instead of relying on the routing tables of intermediate nodes. DSR supports multiple routes to the same destination.

To determine a route to a destination, the source broadcasts a Route Request packet, which contains its address and a sequence number. Intermediate nodes examine these packets to see if they are already listed in the route record or if they've already received this Route Request. If so, it is discarded. If not, the intermediate node appends its address to the route record and broadcasts it. When the Route Request reaches the destination, that node sends a Route Reply message which contains the route record back to the source, where it is cached.

When a node discovers that a link has failed in the course of forwarding data, it sends a Route Error message back to the source node, indicating that the desired route is unavailable. The source may then try another route to the destination if it knows of one, or it may send a new Route Request message.

Intermediate nodes cache any useful information that they forward, including route discovery and route error messages.

Appendix B

Implementation

The bootstrapping algorithm proposed in Chapter 3 and the credit–based incentive scheme proposed in Chapter 5 have been implemented as Java applications using the Microsoft Mesh Networking Academic Resource Toolkit [64] for multihop communication. The toolkit implements an ad hoc routing protocol that uses link quality measurements. It also implements a virtual network adapter that enables higher level software to run unmodified over the ad hoc network.

It is not practical to test large MANET networks, so no specific performance data was measured and instead the focus was on proving the feasibility and correctness of the algorithms for small networks. A discussion of the implementation follows.

B.1 Bootstrap Implementation

The bootstrapping application uses the multicast algorithm presented in [43], which will not be discussed here. It is designed to be used in conjunction with multiple overlays, such as the incentivized P2P application implementation discussed in Section

B.2. The bootstrap application monitors the query and response messages in the P2P–MANET and indicates the best peers that the P2P application should connect to[1].

The bootstrap algorithm is implemented in several Java classes [52], of which we now describe the main ones. The Client class is used initiate join queries, the Server class is used to handle all incoming mesages, and the SendQuery class is used by both the Client and Server classes to transmit packets. It takes care of the details of determining whether to multicast or unicast the packet. Packet exchange between nodes is done via UDP.

The Client class is responsible for sending out join queries. This thread will first look in the host's cache table to see if it already knows of some nodes that are members of the desired overlay. If it has knowledge of an overlay member from previous responses then the Client will send a unicast message to the peer for a join request. If the overlay it is looking for has no entries in the table or there are too few peers a join request message is multicast to the group. This join request message contains the message header, TTL, overlay name, packet sequence number, peer score, and source IP address.

The message header specifies the type of message and can be either a multicast message to all peers in the MANET, or to a specific peer directly. The TTL field is initialized to nine when the packet first leaves the host and is decremented after every hop. If the TTL is equal to zero the intermediate node will not prepare the packet for multicasting and will instead drop it. The overlay name is used to describe the type of the overlay the client is looking for. The packet sequence number is used to identify packets from a particular node. The peer score enables nodes to evaluate

[1]Thanks to Pandeli Kolomitro for his assistance in developing this application

each other and is essentially a utility value. The Source IP address is the address of the node that first generated the packet; this address remains the same for the entire life span of the packet.

The Server class is used to make decisions about incoming and outgoing packets, provide service to other nodes, and decide what peers to connect to based on the score they provide. Once the Server receives a packet it will extract the information from it. Next, the sequence number of the packet that was received is checked and if it has been received before, the packet is dropped. Finally, based on the header value, the packet will be processed accordingly. If it is a multicast packet, the packet will be sent according to the multicast algorithm. Next, the Server checks with the attached application if it is an overlay member and if it is able to allow the requestor to join it as a neighbour. If so, a response is sent back to the sender, and at the same time to all the other nodes in the network.

If the packet header indicates that this is a join query reply message, it is multicasted back out. In addition, the new peer is added to the cache table. If the current node is an overlay member, it checks to see if the responding peer is better than a current neighbour. If so, a message is passed to the attached application and a new connection is suggested. Furthermore a join unicast message is sent back to the potential neighbour.

If the message is a join request, the attached application is consulted to see if there is an available neighbour slot, or if the new peer is better than one of the current ones. If that is the case, the new peer is added. If the conditions are not met a response is sent back indicating that no slots are available.

A peer table is used to organize peers that are part of the same overlay. The peer

table stores the maximum number of peers that the attached application can connect to and sorts peers based on their score. This table is updated every time a new peer joins or leaves the overlay. If a new peer is added to the table and the maximum number of neighbours is not reached, or the new entry is a better one, a message will be sent to the attached application to indicate that this new peer should become a neighbour.

B.2 Incentivized P2P Implementation

The credit–based incentive pricing scheme was implemented as an incentivized P2P file–sharing Java application [56] and was designed to be used together with a bootstrapping application discussed in Section B.1. The incentivized P2P application creates a Gnutella–like unstructured overlay with the assistance of the bootstrapping application. It allows users to query for files and determines what prices to charge for sharing and forwarding, based on the scheme discussed in Section 5.3. Users are awarded credits both for uploading files and for forwarding data, and use these credits when downloading files. The content distribution scheme is a simple file transfer between the client and the selected server, with credits flowing to the intermediate nodes [2].

Joining the P2P overlay is accomplished through the use of the bootstrapping implementation. The P2P application registers with the bootsrapping service by providing a unique overlay name and subsequently executing it in a separate thread. The applications use an agreed upon interface to communicate. The bootstrap application is provided with two methods, one to add a peer and one to replace a peer. Each

[2]Thanks to Paul Lanyon for his assistance in developing this application

peer maintains a list of up to five peers they are currently connected to. Neighbours maintain a persistent TCP connection, using it to forward all data. If the bootstrap application discovers a "better" peer, its IP address, along with the peer it should be replacing, is passed up to the P2P program. At this point the P2P application will disconnect from the current neighbour as long as no data is being sent and add the new peer. In this manner the two applications exist in a layered architecture. The bootstrap application handles the lower–level discovery of peers while the P2P application runs on top, handling all the information at the application level. Once a single IP address is passed up to the application, the user may begin using the file–sharing program.

There are essentially four types of messages, each with a corresponding response: Connection Request, Ping, Query, and Download Request. The first part of each message is an integer, indicating the type of message. The second part of the message always contains the sender's IP address and the remaining data varies depending on the type of message.

Connection Request messages are used to establish a semi–permanent TCP connection between two nodes. When a node receives a new IP address from the bootstrapping application running below it, it will decide whether it wants to add that node as a peer or not. If there is an empty space in it's connection list (which has room for five peers), it will automatically add the peer. If its list is full, it will add the peer if it is notified of another it should replace. When a node receives a connection request it can either respond to or ignore the request. If it wants to add the requesting peer to its list, the node responds with a Connection Accept message, returning its IP address and subsequently adds the peer to its connection list. When the node that

initiated the communication receives the Connection Accept message, the new peer is added to the connection list as well. If no message is returned the TCP connection will be closed and the node will not be added.

Pings are messages sent between two peers at a regular interval to ensure they are still communicating properly. The default time is every thirty seconds. When a node receives a Ping from one of its peers, it responds with a Pong that includes its IP address. If the node that sent the Ping does not receive a response within a certain interval (3 seconds by default), it marks the non-response in a table. If the peer does not respond to a second Ping, it is removed from the connection list, as it is determined that either the peer has left the network or is otherwise incapacitated.

The purpose of Query messages is to determine which nodes have a desired file. A Query message contains a unique query ID (based on the user's IP address to ensure uniqueness), as well as the query string, which is the text a user wishes to match to file names. Queries are flooded over the network when the initial node sends the message to each of its peers. Each peer checks to see if it has already received a query containing the query ID. If it has, it does nothing with the Query message. If it has not, the ID is added to a list, the node concatenates its IP address to the end of the Query message, and it is forwarded to every peer in its list. After forwarding the message, the node checks the query string against its local file list. If a match is found, the node creates a Query Response message that includes the file name, file size and its IP address. An estimated delay and the cost the node wishes to charge for the file is added to the Query Response message. Once the new message is created, the node sends it backwards along the same path the Query message took. Each subsequent node along the path adds a cost and estimated delay to the message

before forwarding it. When the Query Response message reaches the initial node, the message will contain the path the Query took as well as the total cost of the file and an estimated delay.

The Download Request actually involves two different types of messages. The first is sent over a new TCP connection to the server node. It starts with a Download Request indicator, followed by the source IP address, then the name of the file requested, and finally the number of credits being transferred between the two nodes. The second is a Credit Transfer message that is sent to each node along the path the Query took. When the server node receives the Download Request message, it checks whether it is currently transferring any other files. If the number of current transfers is equal to the maximum number of allowable transfers, the file is queued; otherwise, the server begins loading the file into a buffer. The server then responds with a Download Accepted message, which simply acknowledges that it will be transferring the file, while sending the file size again so that the client can reserve enough space. Following this, the server begins transferring the file over the newly created TCP connection. Once the file has completed transferring, the two nodes close the connection and the new file is added to the client's local file list.

The application is implemented with several classes, of which the most important will now be discussed.

The BookKeeper class maintains all information relevant to the incentive scheme in one location, providing easy access to costs and delay estimates. It keeps track of the number of packets being sent and received, as well as the number of queries and downloads received. The class stores all of this information in containers that mark the time they were received. A vector containing instances of the FileEntry

class is also maintained; these instances contain information on specific files. The BookKeeper class provides methods to determine the cost and delay of each file matching any given query.

Connection is the main class used for communication with peers. It maintains a TCP socket connection with a single peer and the Node class holds each instance of Connection in an array. Each instance runs in its own thread, looping in the main method to read any strings sent over the socket. When a string is read, the message indicator is checked and the remainder of the string is then passed on to the appropriate method. The Connection class also provides a method to send each type of message. A string, or an instance of a class, is passed and the Connection class creates a message in the appropriate format, subsequently writing it to the socket.

When a user wishes to download a file, a new instance of the Downloader class is created, passed an instance of the QueryResponse class in its constructor, and started in a new thread. All the information regarding the desired file can be found in the QueryResponse class, including the server IP address, the name of the file and the file size. Downloader then opens a new socket connection to the server, sends a download request and waits until the server sends a Download Accepted message. The class then enters a loop, reading in bytes over the socket until the file is completely transferred, at which time the thread closes.

The Listener class simply runs in a thread and listens for incoming TCP connections. It contains the single Server Socket that the P2P program uses. Whenever it receives a new connection it creates an instance of the Connection class and returns to its primary function.

Node is the main class. It is initiated when the user runs the P2P application.

It creates instances of all the other classes and is the mediator for all data passed throughout the system. It creates the local file list and starts all the required timers for checking connections, shifting data in the BookKeeper class and updating the file list. It holds the list of Connection classes, creates the GUI and keeps track of all queries sent and received. It deals with query responses, initiating downloads and sends, as well as maintaining system information. The Node class has a hand in almost every aspect of the application.

The Query class maintains information specific to each query the user sends. It contains the query ID, the query string and all responses. Each instance of the Query class is stored in a vector inside Node. Each instance of Query also maintains a list of responses, with each response being stored inside the QueryResponse class. The QueryResponse class contains the matching file name and file size, the path the query took, the total cost and the total estimated delay.

Each instance of the Sender class is responsible for sending a single file. The file name and a socket are passed in the constructor and the class is run in a new thread. The file to be sent is read with a FileInputStream and then written to the socket. After the file is finished transferring the thread is closed. SendManager maintains the queue of Sender classes. When a download is requested, the Sender is passed to SendManager. If the queue is empty, Sender is started, otherwise it is added to the end of the queue and waits its turn.

B.3 Test Scenarios

We tested the two applications together under different network configurations. The tests were successful and the results met with our expectations. The equipment used

Table B.1: Devices used in testing

IP Address	Device Description
10.1.1.11	Toshiba Tecra M2 Laptop PC Intel Pentium M 1.6 GHz 512 MB RAM Intel PRO 2200BG Wireless
10.1.1.12	Toshiba Tecra Laptop PC Intel Pentium 4 1.4 GHz 256 MB RAM Intel PRO 2200BG Wireless
10.1.1.15	ACF COMPAQ Presario 900 Desktop PC AMD Athlon XP1800+ 533 MHz 256 MB RAM Linksys Instant Wireless Network PC Card V3.0
10.1.1.17	ACER Aspire 3000 Desktop PC AMD Sempron 2800+ 533 MHz 1 GB RAM Broadcom 802.11g Wireless
10.1.1.18	Custom Built Desktop PC Intel P4 1.5 GHz 512 MB RAM D–Link Air Plus Xtreme G DWL–G520
10.1.1.19	Custom Built Desktop PC Intel P4 2.0 GHz 1 GB RAM D–Link WDA–1320 PCI Card

during the tests are shown in Table B.1 along with the associated IP addresses to serve as identifiers. All machines were running Windows XP SP 2.

B.3.1 Bootstrapping

Three network configurations were used to test the bootstrapping application by itself and verify its functionality. First, a short line configuration, with the routing tables on the devices were configured as shown in Figure B.1. Node A is the client node and

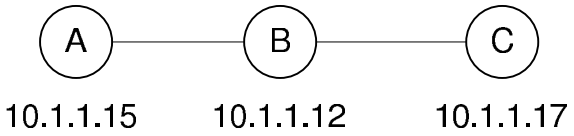


Figure B.1: Short Line Configuration

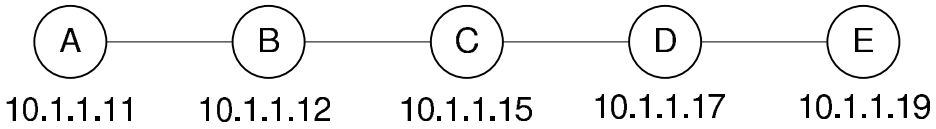


Figure B.2: Long Line Configuration

node C is the server node. Node A sent a multicast query and node C responded with multicast response information. After this phase, node A and node C exchanged a unicast confirmation and acknowledgement.

Next, a longer line configuration, shown in Figure B.2 was tested. Node A is the client node and Node E is the server node. All packets must go through four hops to reach the destination. This configuration tested the performance over multiple hops, and the results proved to be successful.

Finally, a multi–node network configuration was tested to verify that the bootstrapping service could find multiple peers and choose the best neighbours (see Figure B.3). This test requires the P2P file–sharing application in order to check the results. Node B is the client node and nodes A, E, F, C, and D run the same service. The nodes start joining the P2P–MANET one at a time and all have the same initial score. The P2P application is set up to connect to only the three best peers at a time. As nodes communicate over the MANET, their energy level gradually decreases. Assuming that all peers start at the same energy level, the node that joins the network last has the highest energy level, and hence the highest score. In our configuration, node A was the one to join the network first and node C joined the network last. The test

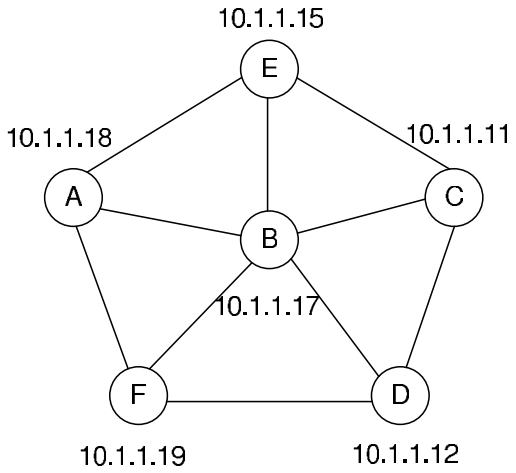


Figure B.3: Multi–Node Configuration

output showed that the peers performed as expected. Node C was listed as the one with the highest score in the peer table, and replaced node A which had the lowest score.

A GUI was implemented for the bootstrap application in order to verify its functionality. Figure B.4 shows the GUI as the application starts up. Once the user clicks the "Join WMN" button, the device joins the local multicast group. Figure B.5 shows the user entering a search query after having clicked the "Join Service Group" button. The user is searching for the "P2P" overlay. Figure B.6 shows the results of the query appearing in a new tab. Four peers have responded and they are ranked by utility value, or score. The application was designed with the idea of multiple P2P overlays existing, and this is shown in Figure B.7. A gaming overlay, a P2P file–sharing overlay, and a user sharing a printer have been found.

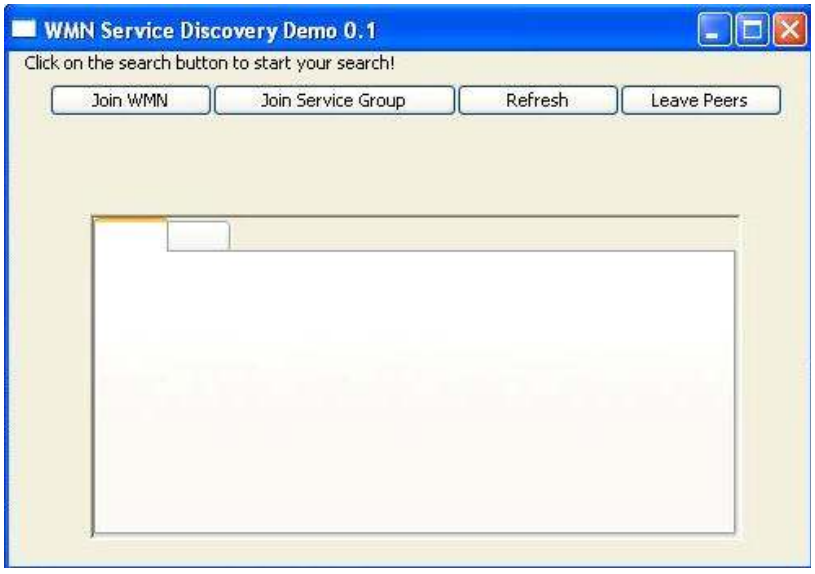


Figure B.4: The GUI on startup

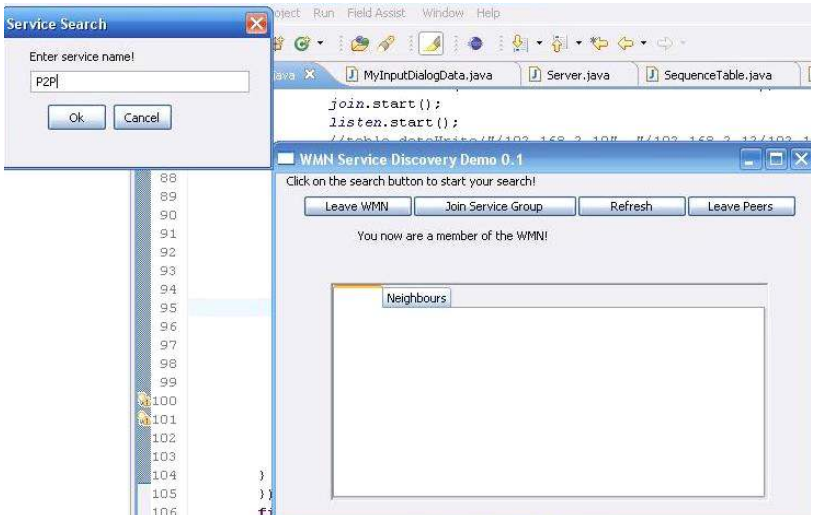


Figure B.5: Searching for an overlay

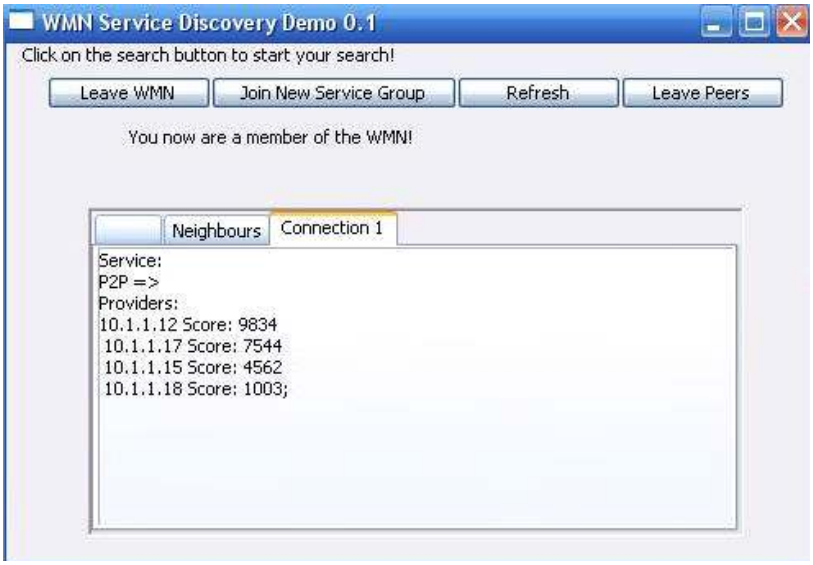


Figure B.6: Query responses

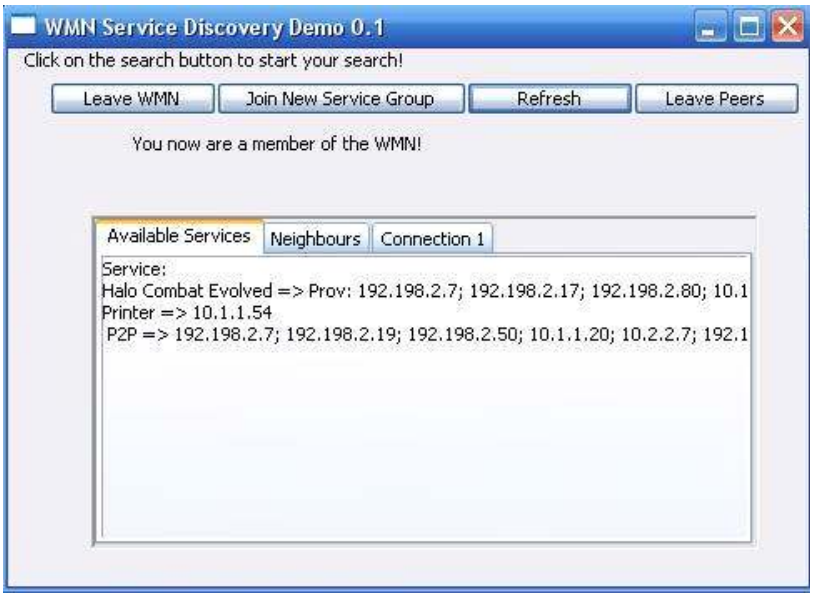


Figure B.7: Multiple overlays

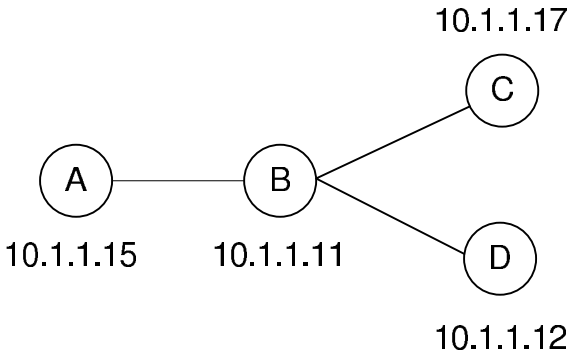


Figure B.8: Test scenario 1

B.3.2 Incentivized File–Sharing

The file–sharing application was tested under three different test scenarios to measure the incentive scheme and application behaviour on the MANET. A description of the scenarios follows.

In the first test scenario, the network was configured as shown in Figure B.8. A file of size 3.26 MB was placed on both nodes C and D. The purpose of this scenario was to test the costs calculated by the incentive scheme. Node A was used to send a Query message for the file, which both C and D would respond to. All variables on these two machines were set to the same initial values. Each charged the same price of 37 credits for the first query. For the second query, both charged a price of 56, confirming that the price was raised after receiving queries. After downloading the file from node D and sending out another query, the price was set to 154 from D and 84 from C, as shown in Figure B.9. This confirms that peers charge more after receiving both queries and downloads. Furthermore, it was verified that node B received credits, confirming the fact that credits are distributed to intermediate nodes.

In the second scenario the network was configured as shown in Figure B.10. The

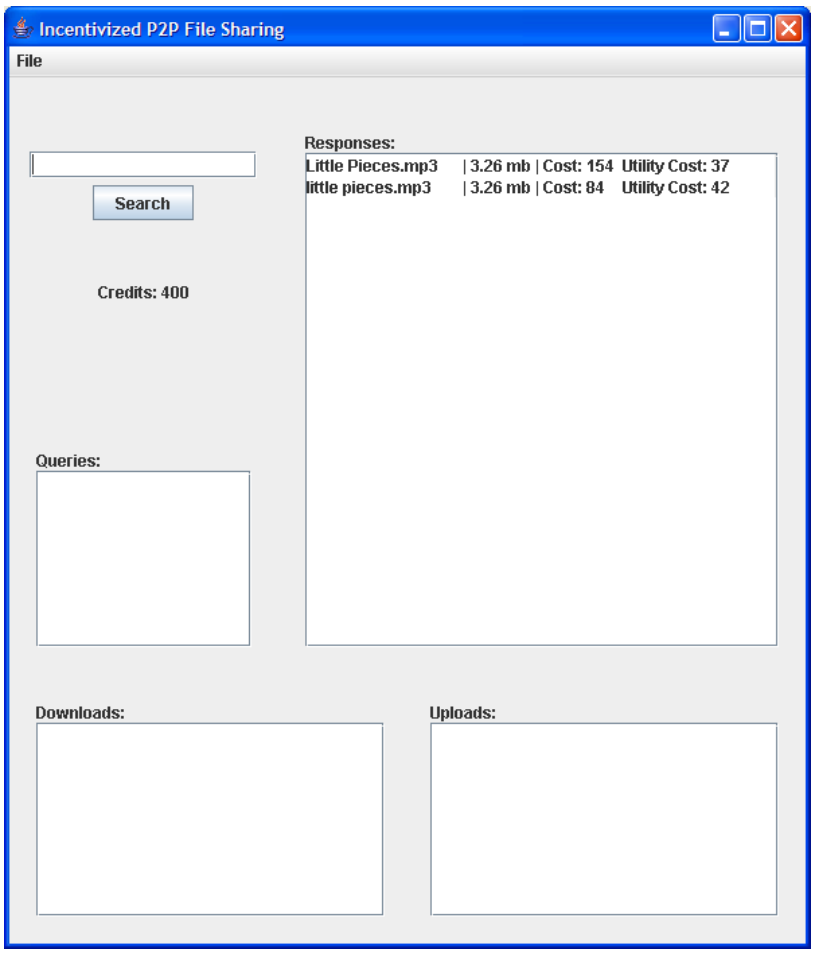


Figure B.9: Screenshot from test scenario 1

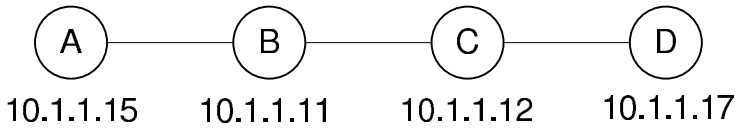


Figure B.10: Test Scenario 2

same file of 3.26 MB was placed on nodes B, C and D. The scenario initially began with only nodes A and B, and the file was transferred between them. Next C was added and the same file was transferred, forcing the transfer through node B. Finally the file was transferred from D to A, forcing it to pass through both intermediate nodes. Each file transfer was timed and the resulting values were:

- Transferring from B to A: 13,159 milliseconds
- Transferring from C to A: 29,822 milliseconds
- Transferring from D to A: 35,301 milliseconds

This test confirmed that multihop transfers take longer due to the need to pass over more hops as well as MAC–layer collisions.

The final scenario was configured as shown in Figure B.11. It was designed to test the moving average functionality of the incentive scheme and involved only two machines. Three queries were sent in succession by node A for a file on node B. The price offered by B rose with each query, since the "popularity" of the file was deemed to be increasing. A fourth query was sent more than five minutes later, as this is the time period in which nodes move queries into historical demand. This final query returned a slightly lower price than the third query, confirming that more recent queries are priced higher, and that B had begun lowering its cost because it had not received a query for that file in some time.

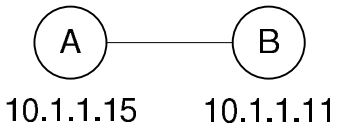


Figure B.11: Test Scenario 3

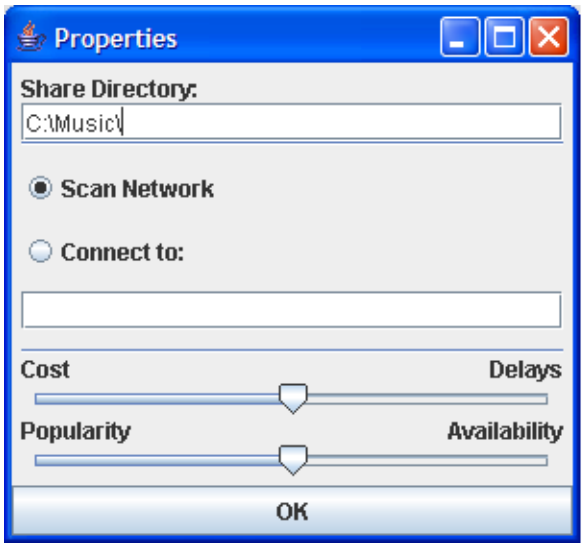


Figure B.12: User options dialog

Figure B.12 shows the application's options dialog, which allows the user to set their preference parameters as a client (cost vs. delay) and as a server (popularity vs. availability).