EFFICIENT POWER-AWARE DATA ACCESS

IN MOBILE ENVIRONMENTS

A Thesis in
Computer Science and Engineering

by
Liangzhong Yin

© 2004 Liangzhong Yin

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

Sept. 2004
We approve the thesis of Liangzhong Yin.

Date of Signature

______________________________
Guohong Cao
Associate Professor of Computer Science and Engineering
Thesis Adviser
Chair of Committee

______________________________
Chita R. Das
Professor of Computer Science and Engineering

______________________________
Wang-Chien Lee
Associate Professor of Computer Science and Engineering

______________________________
Aylin Yener
Assistant Professor of Electrical Engineering

______________________________
Raj Acharya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering
Abstract

In the last decade, there has been tremendous growth in both wireless communications and the Internet. These two technologies are making ubiquitous computing possible. We can envision the scenario of the near future where wireless data access will be readily available to the public anywhere and anytime. Before this vision becomes a reality, however, some issues still need to be resolved. Because of the physical limitations of the wireless network and devices, such as limited bandwidth, battery power, and computation power, it is much more important to design efficient data access schemes in wireless environments than in wired environments. This thesis addresses these issues by designing data access schemes in different wireless environments to improve their performance.

We first address the data access issue in single-hop based mobile environments, where mobile nodes access data through a base station that is one hop away. In this environment, the base station usually adopts the broadcast technique to disseminate data to mobile nodes. As data are available on the broadcast channel, mobile nodes may prefetch them to reduce query latency. However, prefetch consumes valuable battery power and hence should be performed carefully. Therefore, we propose a value-based prefetch scheme to determine which data to prefetch. Then adaptive prefetch schemes are designed so that the prefetch can be done while adapting to mobile nodes’ power levels to achieve a balance between query latency and power consumption.
When there is no pre-installed base station, mobile nodes may form a multi-hop based wireless network and forward packets for each other. As long as one mobile node maintains a database or connects to the Internet, other mobile nodes can still access data through multi-hop links. In this environment, we propose to use two techniques to improve data access efficiency: cooperative caching and data replication.

Cooperative caching fully exploits the potential of caching by serving requests for other mobile nodes besides the cache owner. In mobile environments, a cooperative caching scheme should adapt to the mobile network where mobile nodes have limited computation and communication power, and where the network topology changes frequently. We propose several schemes that are suitable for mobile environments. These schemes are able to reduce query latency and power consumption without incurring high overhead.

Multi-hop based mobile environments have a high link/node failure rate, and thus the data access may be affected when the requested data item is not accessible because of network partitions. Data replication schemes are studied in this thesis to address this issue. Existing schemes usually focus on either reducing the query delay or improving data accessibility, but fail to consider the tradeoffs between the two performance metrics. The data replication schemes proposed in this thesis show that these two metrics are closely related and balance the tradeoffs between them.

Although this thesis studies different mobile environments, the focus is on efficient data access to reduce power and other resource consumption. We show that through intelligent use of the local cache, either by prefetching, cooperative caching, or data replication, mobile nodes are able to access data more efficiently. We hope that these
research results can help build a better wireless environment for future mobile users. We also hope that other researchers find our analytical and simulation methods helpful so that new and more exciting results can be generated.
# Table of Contents

List of Tables ......................................................... xi

List of Figures ......................................................... xii

Acknowledgments ...................................................... xiv

Chapter 1. Introduction ............................................. 1
  1.1 Single-hop and Multi-hop Based Wireless Data Access Models .... 1
  1.2 Thesis Overview ............................................... 3
    1.2.1 Data Access in Single-hop Based Mobile Environments .... 3
    1.2.2 Data Access in Multi-hop Based Mobile Environments .... 4
      1.2.2.1 Cooperative Caching in Multi-hop Based Mobile Environments ....................... 5
      1.2.2.2 Data Replication in Multi-hop Based Mobile Environments ......................... 6
  1.3 Contributions .................................................... 7
  1.4 Thesis Outline ................................................... 8

Chapter 2. Data Access in Single-hop Based Mobile Environments .... 10
  2.1 Introduction ..................................................... 10
  2.2 Preliminaries ..................................................... 13
    2.2.1 The System Model .......................................... 13
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.2</td>
<td>The Cache Invalidation Model</td>
<td>13</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Performance Metrics</td>
<td>14</td>
</tr>
<tr>
<td>2.3</td>
<td>The Adaptive Value-based Prefetch (AVP) Scheme</td>
<td>15</td>
</tr>
<tr>
<td>2.3.1</td>
<td>The Value-based Prefetch (VP) Scheme</td>
<td>16</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Analysis of the Value Function</td>
<td>18</td>
</tr>
<tr>
<td>2.3.3</td>
<td>Parameter Estimation</td>
<td>22</td>
</tr>
<tr>
<td>2.3.4</td>
<td>The Adaptive Value-based Prefetch (AVP) Scheme</td>
<td>23</td>
</tr>
<tr>
<td>2.3.4.1</td>
<td>The Value of $N_p$</td>
<td>24</td>
</tr>
<tr>
<td>2.3.4.2</td>
<td>AVP$_T$: Adapting $N_p$ to Reach a Target Battery Life</td>
<td>24</td>
</tr>
<tr>
<td>2.3.4.3</td>
<td>AVP$_P$: Adapting $N_p$ based on the Power Level</td>
<td>25</td>
</tr>
<tr>
<td>2.4</td>
<td>Performance Evaluation</td>
<td>27</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Simulation Model and Parameters</td>
<td>27</td>
</tr>
<tr>
<td>2.4.1.1</td>
<td>The Client Model</td>
<td>27</td>
</tr>
<tr>
<td>2.4.1.2</td>
<td>The Server Model</td>
<td>28</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Simulation Results: VP</td>
<td>30</td>
</tr>
<tr>
<td>2.4.2.1</td>
<td>The Effects of the Mean Update Arrival Time</td>
<td>30</td>
</tr>
<tr>
<td>2.4.2.2</td>
<td>Comparisons with Simple Prefetch Schemes</td>
<td>33</td>
</tr>
<tr>
<td>2.4.2.3</td>
<td>The Effects of the Cache Size</td>
<td>33</td>
</tr>
<tr>
<td>2.4.2.4</td>
<td>The Effects of the Zipf Parameter $\theta$</td>
<td>36</td>
</tr>
<tr>
<td>2.4.2.5</td>
<td>The Effects of the Access Pattern Variation</td>
<td>38</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Simulation Results: Adaptation</td>
<td>38</td>
</tr>
<tr>
<td>2.4.3.1</td>
<td>Adapting $N_p$ to Reach a Target Battery Life</td>
<td>38</td>
</tr>
<tr>
<td>2.4.3.2</td>
<td>The Adaptation of $N_p$ based on the Power Level</td>
<td>42</td>
</tr>
</tbody>
</table>
Chapter 3. Data Access in Multi-hop based Mobile Environments: Cooperative Caching

3.1 Introduction ................................ 46

3.2 Proposed Basic Cooperative Cache Schemes ..................... 49
  3.2.1 System Model ................................ 49
  3.2.2 Cache the Data (CacheData) ............................ 50
  3.2.3 Cache the Data Path (CachePath) ...................... 51
  3.2.4 Performance Analysis ............................... 54

3.3 A Hybrid Caching Scheme (HybridCache) ......................... 60
  3.3.1 Cache Replacement Policy ............................. 61

3.4 Performance Evaluation .................................. 64
  3.4.1 The Simulation Model ................................. 65
  3.4.2 Simulation Results: HybridCache ....................... 68
    3.4.2.1 Fine-tuning CachePath ........................... 68
    3.4.2.2 Fine-tuning HybridCache ......................... 69
    3.4.2.3 Effects of the Cache Size ......................... 70
    3.4.2.4 Effects of the Query Generate Time .............. 73
    3.4.2.5 Effects of the Zipf Parameter $\theta$ ........... 74
    3.4.2.6 Effects of TTL ................................. 75
    3.4.2.7 Effects of the Node Density .................... 76
3.4.3 Simulation Results: Comparisons .......................... 77
  3.4.3.1 Effects of the Cache Size .............................. 78
  3.4.3.2 Effects of the Mean Query Generate Time $T_{query}$ . 80
3.4.4 Simulation Results: Cache Replacement Policy .............. 81
3.5 Related Work .................................................................. 83
  3.5.1 Caching Schemes in Wired Networks ......................... 83
  3.5.2 Caching Schemes in Wireless Networks ...................... 84
  3.5.3 Cache Replacement Policies ................................. 86
3.6 Summary of Cooperative Cache Schemes ......................... 87

Chapter 4. Data Access in Multi-hop based Mobile Environments: Data Replication ................................................. 88
4.1 Introduction .................................................................. 88
4.2 Related Work .................................................................. 90
4.3 Preliminaries .............................................................. 92
  4.3.1 System Model ..................................................... 92
  4.3.2 Problem Analysis ............................................... 93
4.4 The Proposed Data Replication Schemes ......................... 95
  4.4.1 An Example ....................................................... 95
  4.4.2 Heuristics .......................................................... 97
  4.4.3 Data Replication Schemes ..................................... 98
    4.4.3.1 Greedy Schemes ............................................ 98
    4.4.3.2 The One-To-One Optimization (OTOO) Scheme ... 101
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Simulation parameters</td>
<td>29</td>
</tr>
<tr>
<td>3.1</td>
<td>Simulation parameters</td>
<td>67</td>
</tr>
<tr>
<td>4.1</td>
<td>Access Probability to Data Items</td>
<td>96</td>
</tr>
<tr>
<td>4.2</td>
<td>Simulation parameters</td>
<td>108</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The data access cost model</td>
<td>17</td>
</tr>
<tr>
<td>2.2</td>
<td>The AVPₜ scheme</td>
<td>26</td>
</tr>
<tr>
<td>2.3</td>
<td>Performance under varying update arrival time</td>
<td>31</td>
</tr>
<tr>
<td>2.4</td>
<td>Comparisons with simple prefetch schemes</td>
<td>34</td>
</tr>
<tr>
<td>2.5</td>
<td>Performance as a function of the cache size</td>
<td>35</td>
</tr>
<tr>
<td>2.6</td>
<td>The effects of the Zipf parameter θ</td>
<td>37</td>
</tr>
<tr>
<td>2.7</td>
<td>The effects of the access pattern variation</td>
<td>39</td>
</tr>
<tr>
<td>2.8</td>
<td>AVPₜ: adapting ( N_p ) to reach a target battery life</td>
<td>40</td>
</tr>
<tr>
<td>2.9</td>
<td>The energy level and system performance as a function of time</td>
<td>43</td>
</tr>
<tr>
<td>3.1</td>
<td>An ad hoc network</td>
<td>50</td>
</tr>
<tr>
<td>3.2</td>
<td>Performance comparison</td>
<td>58</td>
</tr>
<tr>
<td>3.3</td>
<td>The hybrid caching scheme</td>
<td>62</td>
</tr>
<tr>
<td>3.4</td>
<td>Fine-tuning CachePath</td>
<td>68</td>
</tr>
<tr>
<td>3.5</td>
<td>Fine-tune HybridCache</td>
<td>69</td>
</tr>
<tr>
<td>3.6</td>
<td>The system performances as a function of the cache size</td>
<td>71</td>
</tr>
<tr>
<td>3.7</td>
<td>The average query delay as a function of the mean query generate time ( T_{query} )</td>
<td>73</td>
</tr>
<tr>
<td>3.8</td>
<td>The average query delay as a function of ( θ )</td>
<td>75</td>
</tr>
<tr>
<td>3.9</td>
<td>The average query delay as a function of TTL</td>
<td>76</td>
</tr>
<tr>
<td>3.10</td>
<td>The average query delay under different node density</td>
<td>77</td>
</tr>
</tbody>
</table>
3.11 The performance as a function of the cache size .......................... 79

3.12 The performance as a function of the mean query generate time $T_{\text{query}}$ 80

3.13 The performance as a function of the cache size .......................... 82

4.1 Data accessibility under different link failure probability between $N_1$ and $N_2$ ................................................................. 97

4.2 The worst case data accessibility of greedy scheme compared to the super-optimal scheme under different memory size $C$ (the total number of data items $n = 100$) .................................................. 100

4.3 Performance of RN as a function of $T_r$ when nodes have the same access pattern ................................................................. 109

4.4 Performance as a function of $\theta$ when nodes have different access pattern 110

4.5 Performance as a function of $\theta$ when nodes have the same access pattern 111

4.6 Performance as a function of the radio range when nodes have different access pattern ................................................................. 113

4.7 Performance as a function of radio range when nodes have the same access pattern ................................................................. 115

4.8 Performance as a function of memory size when nodes have different access pattern ................................................................. 116

4.9 Performance as a function of memory size when nodes have the same access pattern ................................................................. 116

4.10 Performance as a function of $\delta$ when nodes have different access pattern 117

4.11 Performance as a function of $\delta$ when nodes have the same access pattern 118
Acknowledgments

I am most grateful and indebted to my advisor, Professor Guohong Cao for his support throughout my thesis research. He not only created a motivating, enthusiastic, and critical research environment, but also provided valuable research experience and expertise. Professor Cao guided me through the countless problems and setbacks I met in my research. Without all his guidance and help, this thesis would not have been possible. It has been a great honor and pleasure for me to conduct this thesis under his supervision. I would also like to thank Professors Chita Das, Wang-Chien Lee, and Aylin Yener for spending their time on my committee and for providing insightful and constructive comments on this thesis.

During my time at Penn State, I met many friends, too many to list here. They helped me to develop new ideas through discussion and criticism. They also helped me to have a good time besides just doing research at Penn State. I have been really lucky to have met all of them.

Last but not least, I want to thank my wife, who always supported me in good and hard times. Her love and care made my life meaningful and fun. I’m also thankful to my other family members. Although they live far away, their love and support are always with me.
Chapter 1

Introduction

The falling cost of both communication and mobile devices (laptop computers, personal digital assistants, hand-held computers, etc.) has made it possible for mobile users to access various kinds of services at any time and any place. However, existing wireless services are limited by constraints such as narrow bandwidth, frequent disconnections, and short battery lifetime. Thus, mechanisms to efficiently disseminate information to mobile clients have received considerable attention [5, 9, 11, 28, 59].

To study data access in the mobile environment, we need to understand the different models of data access in this environment. Today, there are different types of wireless networks, and even more kinds of mobile devices. The various types of data access scenarios in mobile environments can be classified into two categories: the single-hop based data access model and the multi-hop based data access model, which are discussed in the following sections. This thesis studies the data access in both models and proposes schemes that enable mobile nodes to access data more efficiently, in terms of power consumption, query latency, etc.

1.1 Single-hop and Multi-hop Based Wireless Data Access Models

In wireless environments, mobile nodes are either one hop away from the server or not directly covered by the server because of limited radio range. Mobile nodes that are
Mobile nodes that are not directly covered by the server may form a multi-hop based wireless ad hoc network and relay data hop by hop for each other. These two data access models are referred to as the single-hop based wireless data access model and the multi-hop based wireless data access model.

Broadcasting has been shown to be an effective data dissemination technique in the single-hop based data access model by many studies [2, 5, 25, 30]. With this technique, mobile nodes that are covered by the server access data by simply monitoring the broadcast channel until the requested data appear in the channel. Broadcasting exploits the asymmetric nature of the wireless channel, where more bandwidth is available for the downlink, but less is available for the uplink. Furthermore, broadcasting is scalable since the bandwidth consumption is independent of the number of mobile nodes in the system. Despite its advantages, the broadcasting technique has one drawback: it requires pre-installed infrastructures to support the broadcasting networks.

In situations where installing an infrastructure is not possible because it is too expensive or too vulnerable, the multi-hop based data access model may be used. Mobile nodes may form an ad hoc network for data access. Wireless ad hoc networks have received considerable attention because of their potential applications in battlefield, disaster recovery, and outdoor assemblies. Due to the lack of infrastructure support, each mobile node in the ad hoc network acts as a router, forwarding data packets for other nodes. As long as one mobile node is connected to the Internet, all other nodes can still access their required data.
1.2 Thesis Overview

In both single-hop based and multi-hop based data access models, the performance of data access can be improved by utilizing local memory space (local cache): either by caching/prefetching frequently accessed data or by replicating data to mobile nodes. This thesis focuses on how to use the local cache space effectively with different techniques such as prefetching, cooperative caching, and data replication to reduce not only query latency but also resource (battery power, wireless bandwidth, etc.) consumption. The following sections introduce the thesis research in both the single-hop based mobile environment and the multi-hop based mobile environment.

1.2.1 Data Access in Single-hop Based Mobile Environments

In single-hop based mobile environments, data broadcasting has many advantages, but it also introduces some problems. For example, because multiple clients share the downlink channel, they need to wait for their requested data to appear in the channel. This may increase the query latency and the power consumption. One way to alleviate this problem is to cache frequently accessed data on the mobile nodes [5, 11, 67, 72]. In this way, the mobile node can serve many requests from the local cache without sending uplink requests. This not only reduces the mobile node waiting time but also reduces the uplink and downlink messages.

To further reduce the access latency and improve the cache hit ratio, prefetching techniques can be used. Prefetching has many advantages in mobile environments since wireless networks such as wireless LANs and cellular networks support broadcasting.
When the server broadcasts data on the broadcast channel, clients can prefetch interested data without increasing the bandwidth consumption. Note that if the requested data item is not prefetched earlier, the client has to send an uplink request when receiving the query. This not only increases the query delay but also increases the uplink bandwidth requirement. Since the uplink bandwidth is very expensive in wireless networks, prefetching should be used frequently. However, prefetching consumes a large amount of system resources such as battery power on the client side. Although prefetching can make use of the broadcast channel, clients still need to consume power to receive and process the data. Further, in order to prefetch, they cannot power off the wireless network interface. This consumes a large amount of power, even when it is in the idle mode [57]. Since most mobile clients are powered by battery, it is important to prefetch the right data when designing prefetching schemes. Unfortunately, most of the prefetching techniques used in the current cache management schemes [11, 24] do not consider the power constraints of the mobile clients and other factors such as data size, data access rate, and data update rate. To address these issues, we first propose a value-based (VP) scheme, which makes prefetch decisions based on the value of each data item, considering factors such as access rate, update rate, and data size. Then, we extend the VP scheme and present two Adaptive Value-based Prefetch (AVP) schemes, which can achieve a balance between performance and power based on different user requirements.

1.2.2 Data Access in Multi-hop Based Mobile Environments

Most of the previous research [15, 19, 34, 47, 66] in multi-hop based mobile environments focus on the development of dynamic routing protocols that can efficiently find
routes between two communicating nodes. Although routing is an important issue in multi-hop based mobile environments, other issues such as information (data) access are also very important since the ultimate goal of using multi-hop based mobile environments is to provide information access to mobile nodes.

Data access in multi-hop based mobile environments is very challenging because there is no pre-installed infrastructure, and mobile nodes have limited resources. In this environment, requests and data are forwarded hop by hop through wireless links to reach the destination. These wireless links are much less reliable and may incur longer delay than wired links. Therefore, reducing the number of hops that requests and data need to travel can significantly improve the performance of data access in this environment. To achieve this, we propose using two approaches: data caching and data replication.

1.2.2.1 Cooperative Caching in Multi-hop Based Mobile Environments

Mobile nodes can cache their requested data so that requests can be served locally. To fully utilize the cache, mobile nodes can cooperate with each other to serve requests for each other. In this way, more requests can be served by cached data instead of by the distant server. Actually, cooperative caching, which allows the sharing and coordination of cached data among multiple nodes, has been widely used to improve the Web performance in wired networks. These protocols can be classified as message-based, directory-based, or router-based. Wessels and Claffy introduced the Internet cache protocol (ICP) [61], which has been standardized and is widely used. As a message-based protocol, ICP supports communication between caching proxies, using a simple query-response dialog. Directory-based protocols such as cache digests [54] and
summary cache [20] enable caching proxies to exchange information about cached content. The web cache coordination protocol [17], as a router-based protocol, transparently distributes requests among a cache array. These protocols usually assume fixed network topology and often require high computation and communication overhead. Because of resource constraints and node mobility, these techniques may not be applied directly to multi-hop based mobile environments. In this thesis, we design and evaluate cooperative caching techniques to efficiently support data access in multi-hop based mobile environments. We first propose two schemes: CacheData, which caches the passing-by data, and CachePath, which caches the data path. After analyzing the performance of these two schemes, we propose a hybrid approach (HybridCache), which can further improve the performance by taking advantage of CacheData and CachePath while avoiding their weaknesses. Cache replacement policies are also studied to further improve the performance. Both numerical analysis and simulation results show that the proposed schemes can significantly reduce the query delay and message complexity when compared to other caching schemes.

1.2.2.2 Data Replication in Multi-hop Based Mobile Environments

To effectively disseminate data in multi-hop based mobile environments, data replication can also be adopted. Compared to caching, data replication is a proactive approach. It distributes data inside the network and may achieve the following goals:

- By allocating data near the requesting nodes, the query latency can be reduced;
- By duplicating data among the nodes, the data accessibility can be improved.
The data accessibility is defined as the probability that data are accessible when they are requested. In multi-hop based mobile environments, mobile nodes face constant link/node failures. Replicating data can significantly improve the data accessibility compared to keeping data at only one node.

Since both query delay and data accessibility metrics are important for mobile nodes, it is important to address both when designing data replication schemes. Otherwise, schemes that improve one performance metric may hurt the performance of the other. For example, in [26], data replication algorithms that improve the data accessibility are proposed. These algorithms try to distribute data among nodes based on the intuition that neighboring nodes should not hold the same data. This may improve the data accessibility in some cases. However, it has a drawback: a node may have to remove a more frequently accessed data item that is available at neighboring nodes to accommodate another less frequently accessed data item that is not available at neighboring nodes. This may increase the query latency. Moreover, it may actually decrease the data accessibility in some cases because data are not necessarily allocated near the nodes that access them frequently. To deal with these issues, we studied data replication schemes in multi-hop based mobile environments that consider both data accessibility and query latency. The proposed schemes can achieve a balance between these two metrics and provide satisfying system performance.

1.3 Contributions

We have made three major contributions in this thesis.
1. We have studied caching and prefetching in single-hop based mobile environments and proposed power-aware prefetching schemes to improve the data access performance. The proposed Adaptive Value-based Prefetch (AVP) scheme defines a value function which can optimize the prefetch cost to achieve better performance. Also, AVP dynamically adjusts the number of prefetches to get a better tradeoff between performance and power.

2. We have proposed cooperative caching schemes in multi-hop based mobile environments. By caching data and serving requests cooperatively, more queries can be served by a nearby node so that requests and data need to travel fewer hops. This can reduce the query latency and power consumption. The proposed schemes are suitable for resource limited mobile environments because they can improve system performance without incurring a high message overhead.

3. We have also proposed data replication schemes to further improve the performance of data access in multi-hop based mobile environments. Unlike existing research, our schemes consider both query latency and data accessibility. In our research, we studied the tradeoffs between the two metrics and showed that the proposed schemes achieve a balance between them.

1.4 Thesis Outline

The rest of this thesis is organized as follows. In Chapter 2, we present the adaptive power-aware prefetch schemes in single-hop based mobile environments. Then data access schemes in multi-hop based mobile environments are studied: Cooperative
caching schemes are studied in Chapter 3, and Data replication schemes are studied in Chapter 4. Chapter 5 concludes this thesis.
Chapter 2

Data Access in Single-hop Based Mobile Environments

2.1 Introduction

Broadcasting has shown to be an effective data dissemination technique for wireless networks in many studies [2, 5, 30]. With this technique, users access data by simply monitoring the channel until the required data appear on the broadcast channel. To efficiently deliver data on the broadcast channels, content organization and data broadcast scheduling should be based on client access patterns. For example, techniques such as broadcast disks [3] are provided to improve the system performance by broadcasting hot data items frequently. To reduce the client power consumption, techniques such as indexing [30] were proposed to reduce the client tune in time. The general idea is to interleave index (directory) information with data on the broadcast channels such that the clients, by first retrieving the index information, are able to obtain the arrival time of the desired data items. As a result, a client can enter doze mode most of the time, and only wakes up just before the desired data arrive.

Although broadcasting has good scalability and low bandwidth requirement, it has some drawbacks. For example, since a data item may contain a large volume of data (especially in the multimedia era), the data broadcast cycle may be long. Hence, the clients have to wait for a long time before getting the required data. Caching frequently accessed data items at the client side is an effective technique to improve performance
in mobile computing systems. With caching, the data access latency is reduced since some data access requests can be satisfied from the local cache, thereby obviating the need for data transmission over the scarce wireless links. When caching is used, cache consistency must be addressed. Although caching techniques used in file systems such as Coda [55], Ficus [52] can be applied to mobile environments, these files systems are primarily designed for point-to-point communication environment, and they may not be applicable to the broadcasting environment.

Recently, many works [5, 8, 9, 12, 33, 35, 67, 58, 72] have shown that invalidation report (IR) based cache management is an attractive approach for mobile environments. In this approach, the server periodically broadcasts an invalidation report in which the changed data items are indicated. Rather than querying the server directly regarding the validation of cached copies, the clients can listen to these IRs over the wireless channel, and use them to validate their local cache.

To further reduce the access latency and improve the cache hit ratio, prefetching techniques can be used. Prefetching has many advantages in mobile environments since wireless networks such as wireless LANs or cellular networks support broadcasting. When the server broadcasts data on the broadcast channel, clients can prefetch interested data without increasing the bandwidth consumption. Note that if the requested data item is not prefetched earlier, the client has to send an uplink request when receiving the query. This not only increases the query delay but also increases the uplink bandwidth requirement. Since the uplink bandwidth is very expensive in wireless networks, prefetching should be used frequently. However, prefetching consumes a large amount of system resources such as battery power on the client side. Although prefetching can
make use of the broadcast channel, clients still need to consume power to receive and process the data. Further, they cannot power off the wireless network interface, which consumes a large amount of power even when it is in the idle mode [57]. Since most mobile clients are powered by battery, it is important to prefetch the right data when designing prefetching schemes. Unfortunately, most of the prefetch techniques used in the current cache management schemes [11, 24] do not consider power constraints of the mobile clients and other factors such as the data size, the data access rate, and the data update rate. To address these issues, we first propose a value-based (VP) scheme, which makes prefetch decisions based on the value of each data item considering various factors such as access rate, update rate, and data size. As stretch [1, 67] is widely adopted as a performance metric for variable-size data requests, we show by analysis that the VP scheme can indeed achieve the optimal performance in terms of stretch. Then, we extend the VP scheme and present two adaptive value-based prefetch (AVP) schemes, which can achieve a balance between performance and power based on different user requirements. Extensive simulations are provided and used to justify the analysis. Compared to previous schemes, the proposed schemes can reduce the energy consumption and improve the system performance in terms of stretch under various scenarios.

The rest of this chapter is organized as follows. Section 2.2 describes the system model and the performance metrics. In Section 2.3, we present the VP and AVP schemes. Section 2.4 evaluates the performance of the VP and AVP schemes. Related work is provided in Section 2.5. Section 2.6 summarizes this chapter.
2.2 Preliminaries

2.2.1 The System Model

We use a pull-based broadcasting model, which consists of a single server and a number of clients. At the server side, there is a database of $n$ data items: $d_1, d_2, \ldots, d_n$. The server is responsible for maintaining the database and serving the requests of the mobile clients. At the client side, caches are used to save frequently accessed data. When a client needs to access a data item that cannot be found locally, it sends out a query to the server through the uplink channel. On receiving the request, the server sends the reply through the common broadcast channel. Similar to most previous work [9, 11, 67, 72], data can only be updated by the server.

2.2.2 The Cache Invalidation Model

Cached data may become invalid due to server update. To ensure cache consistency, the server broadcasts invalidation reports (IRs) every $L$ seconds. The IR consists of the current timestamp $T_i$ and a list of tuples $(d_x, t_x)$ such that $t_x > (T_i - w \cdot L)$, where $d_x$ is the data item id, $t_x$ is the most recent update timestamp of $d_x$, and $w$ is the invalidation broadcast window size. In other words, IR contains the update history of the past $w$ broadcast intervals. However, any client who has been disconnected longer than $w$ IR intervals cannot use the report, and it has to discard the cached items. Every client, if active, listens to the IRs and invalidates its cache accordingly. To answer a query, the client listens to the next IR and uses it to decide whether its cache is valid or not. Since the client has to wait for the next IR before answering a query, the average
query delay is the sum of the actual query processing time and half of the IR interval. If the IR interval is long, the delay may not be able to satisfy the requirements of many clients.

In order to reduce the query latency, Cao [11] proposed to replicate the IRs $m$ times within an IR interval. As a result, a client only needs to wait at most $\left(\frac{1}{m}\right)^{th}$ of the IR interval before answering a query. Hence, latency can be reduced to $\left(\frac{1}{m}\right)^{th}$ of the latency in the previous schemes. Since the IR contains a large amount of update history information, to save the broadcast bandwidth, after one IR, $m - 1$ updated invalidation reports (UIRs) are inserted within an IR interval. Each UIR only contains the data items that have been updated after the last IR was broadcasted. In this way, the size of the UIR becomes much smaller compared to that of the IR. Although this approach can reduce the query latency when there is a cache hit, clients still need to wait for the data to be delivered if there is a cache miss. To improve the cache hit ratio, the UIR approach also actively prefetch data that are available in the broadcast channel. All the cached data are marked as prefetchable and whenever a data item is broadcasted it will be prefetched if the cached copy has expired. In this thesis, the UIR approach is used for cache management.

2.2.3 Performance Metrics

One widely used performance metric is the response time, i.e., the time between sending a request and receiving the reply. It is a suitable metric for homogeneous settings where different data requests have the same “size”. However, the data requirements of users and applications are inherently diverse, and then to encapsulate all responses into
a single-size broadcast would be unreasonably wasteful. Therefore, unlike some previous work [10, 24], we do not assume that the data items have the same size. When data requests are heterogeneous, response time alone is not a fair measure given that the individual requests significantly differ from one another in their service time, which is defined as the time to complete the request if it was the only job in the system. We adopt an alternate performance measure, namely the stretch [1] of a request, defined to be the ratio of the response time of a request to its service time. The rationale behind this choice is based on our intuition; i.e., clients with larger jobs should be expected to be in the system longer than those with smaller requests. The drawback of minimizing response time for heterogeneous workloads is that it tends to improve the system performance of large jobs (since they contribute the most to the response time). Minimizing stretch, on the other hand, is more fair to all job sizes.

2.3 The Adaptive Value-based Prefetch (AVP) Scheme

The proposed AVP scheme consists of two parts. The first part is the value-based prefetch (VP) scheme, which identifies valuable data items for prefetching. The second part is the adaptive value-based prefetch (AVP) scheme, which determines how many data items should be prefetched.

1Note that in broadcast systems, the service time for a request is the requested data size divided by the bandwidth. For simplicity, we remove the constant bandwidth factor and use the data size to represent the service time.
2.3.1 The Value-based Prefetch (VP) Scheme

In this subsection, we present a value-based function which allows us to gauge the worth of a data item when making a prefetch decision. The following notations are used in the presentation:

- \( n \): the number of data items in the database
- \( \bar{u}_i \): the mean update arrival rate for data item \( i \)
- \( \bar{a}_i \): the mean access arrival rate for data item \( i \)
- \( x_i \): the ratio of update rate to access rate for data item \( i \), i.e., \( x_i = \frac{\bar{u}_i}{\bar{a}_i} \)
- \( p_{a_i} \): the access probability of data item \( i \), \( p_{a_i} = \frac{\bar{a}_i}{n \sum_{k=1}^{n} \bar{a}_k} \)
- \( p_{u_i} \): the probability of invalidating cached data item \( i \) before next access.
- \( l_i \): the access latency for data item \( i \)
- \( f_i \): the delay of retrieving data item \( i \) from the server
- \( s_i \): the size of data item \( i \)
- \( v \): the cache validation delay
- \( D \): the set of all the data items in the database
- \( C_k \): the set of cached data items after the \( k^{th} \) access
- \( U_k \): the set of data items updated between the \( k^{th} \) and the \( (k+1)^{th} \) access
- \( P_k \): the set of data items prefetched after the \( k^{th} \) access

The value function is used to identify the data to be prefetched. Intuitively, the ideal data item for prefetching should have a high access probability, a low update rate, a small data size, and a high retrieval delay. Equation (2.1) incorporates these factors to calculate the value of a data item \( i \).
This value function can be further explained by the data access cost model shown in Figure 2.1. If item $i$ is not in the cache, in terms of the stretch value, it takes $f_i/s_i$ to fetch item $i$ into the cache. In other words, if $i$ is prefetched to the cache, the access cost can be reduced by $f_i/s_i$. However, it also takes $((v + P_{u_i} f_i))/s_i$ to validate the cached item $i$, and update it if necessary. Thus, prefetching the data can reduce the cost by $((f_i - v - p_{u_i} \cdot f_i))/s_i$ for each access. Since the access probability is $p_{a_i}$, the value of prefetching item $i$ is $p_{a_i} (f_i - v - p_{u_i} \cdot f_i)$.

The VP scheme decides which data item should be prefetched based on the value function. The VP scheme is defined as follows. Suppose a client can prefetch $N_p$ data items, the VP scheme prefetches the $N_p$ items which have the highest value based on the
value function. Note that VP is not responsible for determining how many items \((N_p)\) should be prefetched. \(N_p\) is determined by the adaptive scheme, which will be discussed in Section 2.3.4.

### 2.3.2 Analysis of the Value Function

In this subsection, we prove that the value-based prefetch scheme can minimize the access cost given that the number of prefetches is limited. We assume that the arrivals of data update and data access follow Poisson distribution. Then, the inter-arrival time \((t^a_i)\) of data access for item \(i\), and the inter-arrival time \((t^u_i)\) of the update follow exponential distributions with means of \(\overline{a}_i\) and \(\overline{u}_i\). The update event \((U_i)\) for a data item \(i\) during the period from the current time to the arrival time of the next query can be probabilistically written as:

\[
pu_i = \Pr(U) = \Pr(t^u_i < t^a_i) = \int_{t^a_i=0}^{\infty} \int_{t^u_i=0}^{t^a_i} f(t^a_i)g(t^u_i)d_{t^a_i}d_{t^u_i} = \frac{u_i}{a_i + u_i} \tag{2.2}
\]
Therefore the value function can be rewritten as

\[
\text{value}(i) = \frac{p_{ai}}{s_i} (f_i - v - \frac{u_i}{a_i + u_i} \cdot f_i)
\]

\[
= \frac{p_{ai}}{s_i} (f_i \cdot \frac{a_i}{a_i + u_i} - v)
\]

\[
= \frac{p_{ai}}{s_i} (\frac{f_i}{1 + x_i} - v)
\]

(2.3)

We evaluate the access cost by calculating the stretch of the \(k^{th}\) access, which can be defined as:

\[
S_k = \sum_{1 \leq i \leq n} \frac{p_{ai} \cdot l_i}{s_i}
\]

(2.4)

Equation (2.4) can be rewritten considering cache hits and cache misses.

\[
S_k = \sum_{i \in C_k} \frac{p_{ai} \cdot l_i}{s_i} + \sum_{i \in (D-C_k)} \frac{p_{ai} \cdot l_i}{s_i}
\]

(2.5)

In case of a cache hit, there are two cases: cache hit with an up-to-date copy and cache hit with an obsolete copy. Thus, Equation (2.5) can be rewritten as:

\[
S_k = \sum_{i \in C_k} \frac{p_{ai} \cdot l_i}{s_i} \cdot p_{ui} + \sum_{i \in C_k} \frac{p_{ai} \cdot l_i}{s_i} (1 - p_{ui})
\]

\[
+ \sum_{i \in (D-C_k)} \frac{p_{ai} \cdot l_i}{s_i}
\]

(2.6)
The access latency \( l_i \) is equal to \( f_i \) when there is a cache miss. When there is a cache hit, \( l_i = v \) when the cache hit is an up-to-date copy, and \( l_i = v + f_i \) when the cache hit is an obsolete copy. Combining Equations (2.2) and (2.6), we get

\[
S_k = \sum_{i \in C_k} \left( \frac{p_{a_i} \cdot (v + f_i)}{s_i} \cdot \frac{u_i}{u_i + a_i} \right) \\
+ \sum_{i \in C_k} \left( \frac{p_{a_i} \cdot v}{s_i} \cdot (1 - \frac{u_i}{u_i + a_i}) \right) + \sum_{i \in (D - C_k)} \frac{p_{a_i} \cdot f_i}{s_i} \\
= \sum_{i \in C_k} \left( \frac{p_{a_i} \cdot (v + f_i)}{s_i} \cdot \frac{u_i}{u_i + a_i} \right) + \sum_{i \in (D - C_k)} \frac{p_{a_i} \cdot f_i}{s_i} \tag{2.7}
\]

**Theorem 1.** Prefetching items with high value can achieve lower stretch than any other prefetch schemes given that the number of prefetched is limited.

**Proof.** Let \( U_k \) represent the data that have been modified since the last invalidation report and \( P_k \) represent the data that are prefetched after the \( k^{th} \) access. Then, \( C_{k+1} = C_k - U_k + P_k \).

\[
S_{k+1} = \sum_{i \in C_{k+1}} \left( \frac{p_{a_i} \cdot (v + f_i) / (u_i + a_i)}{s_i} \right) + \sum_{i \in (D - C_{k+1})} \frac{p_{a_i} \cdot f_i}{s_i} \\
= S_k + \sum_{i \in U_k} \left( \frac{p_{a_i}}{s_i} \cdot \frac{f_i}{1 + x_i} \right) - v \\
- \sum_{i \in P_k} \left( \frac{p_{a_i}}{s_i} \cdot \frac{f_i}{1 + x_i} \right) - v \tag{2.8}
\]
In Equation (2.8), \( \sum_{i \in U_k} \left( \frac{p_{ai}}{s_i} \left( \frac{f_i}{1+x_i} - v \right) \right) \) cannot be reduced because it is caused by the server update. Equation (2.8) implies that the cost can be reduced by prefetching those items that can maximize \( \sum_{i \in P_k} \left( \frac{p_{ai}}{s_i} \left( \frac{f_i}{1+x_i} - v \right) \right) \). This is exactly what the proposed scheme tries to do: prefetching those items with the maximum sum of value. For any other prefetch scheme that prefetch the same number of data items, let the set of prefetched data items be \( P'_k \) and the cost of the \( (k+1)^{th} \) access be \( S'_{k+1} \), according to the proposed scheme,

\[
\sum_{i \in P_k} \left( \frac{p_{ai}}{s_i} \left( \frac{f_i}{1+x_i} - v \right) \right) > \sum_{i \in P'_k} \left( \frac{p_{ai}}{s_i} \left( \frac{f_i}{1+x_i} - v \right) \right)
\]

Thus

\[
S_{k+1} < S'_{k+1}
\]

This proves that the proposed scheme can minimize the cost among all prefetch schemes that prefetch the same number of data items.

The proposed value-based function is calculated in terms of stretch since the performance metric is stretch. Actually, this value-based function can be easily extended for other performance metrics. For example, if the performance metric is query delay, the value function will be changed to \( value(i) = p_{ai} (f_i - v - p_{u_i} \cdot f_i) \). Similar techniques can be used to prove that this value function can minimize the query delay.
2.3.3 Parameter Estimation

To implement the AVP scheme, we need to estimate parameters $f_i$, $\overline{a}_i$, and $\overline{u}_i$ since they are not constant. To estimate $f_i$, we adopt the exponential aging method, which has been used in TCP to estimate the round-trip delay. It combines both the historical data and the currently measured data to estimate the parameters. Whenever a data item $i$ is fetched from the server, $f_i$ is recalculated as following:

$$f_i = \alpha \cdot f_i^{new} + (1 - \alpha) \cdot f_i$$

where $f_i^{new}$ is the currently measured data retrieval delay, $f_i$ on the right side of the formula is the calculated $f_i$ before the last retrieval of item $i$.

Although this formula can be used to estimate $f_i$, it is not suitable for estimating $\overline{a}_i$ and $\overline{u}_i$ since the access rate and the update rate should be “aged” in the absence of data access. That is, the values of $\overline{a}_i$ ($\overline{u}_i$) should be decreased even if there is no data access (update) over some period of time. We apply techniques, which have been used in [56], to estimate $\overline{a}_i$ and $\overline{u}_i$. This method uses $K$ most recent samples to estimate $\overline{a}_i$ and $\overline{u}_i$ as follows.

$$\overline{a}_i = \frac{K}{T - T_{\overline{a}_i}(K)}$$

$$\overline{u}_i = \frac{K}{T - T_{\overline{u}_i}(K)}$$

where $T$ is the current time, $T_{\overline{a}_i}(K)$ and $T_{\overline{u}_i}(K)$ are the time of the $K^{th}$ most recent access and update. If there are less than $K$ samples, all the available samples
are used to estimate the value. As shown in [56], the best performance can be achieved with small value of K (2 or 3). Thus, the spatial overhead to store these samples is very small. The estimation of $\overline{a_i}$ should be done at the client side since different clients may have different access pattern. However, it is impossible for clients to estimate $\overline{u_i}$ since the data updates occur at the server side. Therefore, the server estimates $\overline{u_i}$ of each data item and piggybacks it to clients when the data item is broadcast.

2.3.4 The Adaptive Value-based Prefetch (AVP) Scheme

Due to limitations of battery technology, the energy available for a wireless device is limited and must be used prudently. If the prefetched data item is not accessed or is invalidated before it is accessed, the energy spent on downloading this item will be wasted. To avoid wasting power, it is important that clients only download the data with high value, but such a strict policy may adversely affect the performance of the system and increase the query delay.

Each client may have different available resources and performance requirements, and these resources such as power may change over time. For example, suppose the battery of a laptop lasts for three hours. If the user is able to recharge the battery within three hours, power consumption may not be an issue, and the user may be more concerned about the performance aspects such as the query latency. However, if the user cannot recharge the battery within three hours and wants to use it a little bit longer, power consumption becomes a serious concern. Since $N_p$ controls the number of data to be prefetched and then affects the tradeoff between performance and power, we propose adaptive schemes to adjust $N_p$ to satisfy different client requirements.
2.3.4.1 The Value of $N_p$

When $N_p$ reduces to 0, there will be no prefetch. As $N_p$ increases, the number of prefetches increases and the power consumption also increases. Since the maximum number of data items to be prefetched is limited by the cache size, $N_p$ is also limited by this number. Intuitively, the query delay decreases as the number of prefetches increases. However, this is not always true considering the overhead to maintain cache consistency. In our cache invalidation model, a client needs to wait for the next IR to verify the cache consistency. This waiting time may increase the query delay compared to the approaches without prefetch. The cost has been quantified in Equation 2.1, where $v$ is the cache invalidation delay. Due to the cost of $v$, the value of a data item may be negative. If $\text{value}(i)$ is negative, prefetching item $i$ not only wastes power but also increases the average stretch. Therefore, $N_p$ should be bounded by $N^{\text{max}}_p$, which is limited by the client cache size and the data value; i.e., a client will not prefetch items with negative values.

The tradeoff between performance and power can be achieved by adjusting $N_p$. In the following subsections, we present two adaptive schemes: the AVP\_T (T for Time) scheme which dynamically adjusts $N_p$ to reach a target battery life time, and the AVP\_P (P for Power) scheme which dynamically adjusts $N_p$ based on the remaining power level.

2.3.4.2 AVP\_T: Adapting $N_p$ to Reach a Target Battery Life

A commuter normally knows the amount of battery energy and the length of the trip between home and office. With these resource limitations, the commuter wants to achieve the lowest query delay. This is equivalent to the problem of adapting $N_p$ to
reach a target battery life and minimize the average stretch. Suppose a battery with $E$ joule lasts $T_1$ seconds when $N_p = N_p^{\text{max}}$, and $T_2$ seconds when $N_p = 0$. It is possible to adjust $N_p$ to reach a target battery life time $T \in [T_1, T_2]$. In AVP-T, the client monitors the power consumed in the past. If it consumes too much power in the past and cannot last $T$ seconds, $N_p$ is reduced. On the other hand, it increases $N_p$ when it found that it has too much power left. Certainly, $N_p$ is bounded by $N_p^{\text{max}}$. Figure 2.2 shows the details of the AVP-T scheme.

### 2.3.4.3 AVP-P: Adapting $N_p$ based on the Power Level

When the energy level is high, power consumption is not a major concern and then trading off energy for performance may be a good option if the user can recharge the battery soon. On the other hand, when the energy level is low, the system should be power-aware to prolong the system running time to reach the next battery recharge time. Based on this intuition, the AVP-P scheme dynamically changes $N_p$ based on the power level. Let $a_k$ be the percentage of energy left in the client. When $a_k$ drops to a threshold, the number of prefetches should be reduced to some percentage, say $f(a_k)$, of the original value. Some simple discrete function can be as follows:

$$f(a_k) = \begin{cases} 
100\% & 0.5 < a_k \leq 1.0 \\
70\% & 0.3 < a_k \leq 0.5 \\
50\% & 0.2 < a_k \leq 0.3 \\
30\% & 0.1 < a_k \leq 0.2 \\
10\% & a_k \leq 0.1 
\end{cases}$$  \hspace{1cm} (2.9)
**Notations:**

- $E$: the amount of initial energy.
- $T$: the target battery lasting time.
- $P_{\text{avg}}$: the estimated average power consumption.
- $T_{\text{old}}, E_{\text{old}}$: the time to obtain the last $P_{\text{avg}}$ and the energy level at that time.
- $T_{\text{new}}, E_{\text{new}}$: the current time and the current energy level.
- $N_{\text{max}}^p$: the maximum $N_p$.

The AVP$_T$ scheme is as follows:

\[
P_{\text{avg}} = \alpha \cdot P_{\text{avg}} + (1 - \alpha) \cdot \left(\frac{E_{\text{old}} - E_{\text{new}}}{T_{\text{new}} - T_{\text{old}}}\right);
\]

\[
\text{if } (|P_{\text{avg}} - E/T|/(E/T) > 0.05 ) \text{ then } \]

\[
N_p = \text{int}(\max(1,N_p) \cdot (1 - 2 \cdot (P_{\text{avg}} - E/T)/(E/T)));
\]

\[
\text{else } \{
\]

\[
\text{if } (P_{\text{avg}} - E/T > 0) \text{ then } N_p + +;
\]

\[
\text{else } N_p - -;
\]

\[
\text{}\}
\]

\[
\text{if } (N_p < 0) \text{ then } N_p = 0;
\]

\[
\text{else if } (N_p > N_{\text{max}}^p) \text{ then } N_p = N_{\text{max}}^p;
\]

Fig. 2.2. The AVP$_T$ scheme
At regular interval, the client re-evaluates the energy level $a_k$. If $a_k$ drops to a threshold value, $N_p = N_p \cdot f(a_k)$. The client only marks the first $N_p$ items in the cache, which have the maximum value, as prefetchable. In this way, the number of prefetches can be reduced to prolong the system running time. Because this is a discrete function, $N_p$ does not need to be frequently updated and the computation overhead is low.

Note that a simple policy which is neither too aggressive nor conservative might result in similar average stretch and lifetime as the VAP scheme if the battery runs out before recharge. However, if the user recharges the battery frequently, this simple policy may not be a good option since it saves power at the cost of delay, but power consumption is not a concern at this time. In contrast, our adaptive scheme tries to tradeoff power for performance at the beginning, and become power-aware when the client cannot recharge in time.

2.4 Performance Evaluation

To evaluate the performance of the proposed methodology, we developed a simulation model similar to that employed in [11, 67]. We compare the value-based prefetch (VP) approach and the adaptive value-based prefetch (AVP) approaches with the prefetch scheme used in UIR [11] under various workload and system settings.

2.4.1 Simulation Model and Parameters

2.4.1.1 The Client Model

Each client generates a single stream of read-only queries. The mean query generate time for each client is $T_{query}$. The access pattern follows a Zipf-like distribution
In the Zipf-like distribution, the access probability of the $i^{th}$ ($1 \leq i \leq n$) data item is represented as follows.

$$p_{i} = \frac{1}{k^\theta \sum_{k=1}^{n} \frac{1}{k^\theta}}$$

where $0 \leq \theta \leq 1$. When $\theta = 1$, it is the strict Zipf distribution. When $\theta = 0$, it becomes the uniform distribution.

### 2.4.1.2 The Server Model

The server broadcasts IRs and UIRs periodically to the clients. The IR/UIR broadcast messages are assigned the highest priority whereas the rest of the messages are of equal priority. This ensures that IRs and UIRs can be broadcast regularly over the wireless channels with the broadcast interval specified by $L$. The IR interval is set at 20 seconds and the UIR is replicated 4 times within each interval. The other messages are served on a first-come-first-serve basis. Should the server be in the middle of a transmission when an IR or UIR has to be sent, the IR/UIR broadcast is deferred till the end of the current packet transmission. There are $n$ data items at the server side. The size ($s_i$) of item $i$ grows linearly from $s_{\text{min}}$ to $s_{\text{max}}$ as $i$ increases. As a result of joint distribution of access pattern and data size, the item with smaller size will be accessed more frequently than bigger ones. This has been commonly observed in traces [6, 23].
The server generates a single stream of updates separated by an exponentially distributed update interval time. The whole database is divided into two subsets: the frequently-updated subset which contains the first 20% of the data items (id from 0 to \( n \times 20\% - 1 \)) in the database, and the rarely-updated subset which contains the rest of the data. 80% of the updates are randomly distributed inside the frequently-updated subset. The rest of the updates are randomly distributed in the rarely-updated subset. It is assumed that the bandwidth is fully utilized for broadcasting IRs and UIRs and serving client requests. The server processing time is considered to be negligible. Most of the system parameters and their default values are listed in Table 2.1.

<table>
<thead>
<tr>
<th>Table 2.1. Simulation parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of data items ((n))</td>
</tr>
<tr>
<td>Number of clients</td>
</tr>
<tr>
<td>(s_{min})</td>
</tr>
<tr>
<td>(s_{max})</td>
</tr>
<tr>
<td>Mean update time ((T_{update}))</td>
</tr>
<tr>
<td>Broadcast interval ((L))</td>
</tr>
<tr>
<td>Broadcast window ((w))</td>
</tr>
<tr>
<td>Broadcast bandwidth</td>
</tr>
<tr>
<td>Uplink bandwidth</td>
</tr>
<tr>
<td>Relative cache size</td>
</tr>
<tr>
<td>Mean query generate time ((T_{query}))</td>
</tr>
<tr>
<td>Zipf distribution parameter (\theta)</td>
</tr>
</tbody>
</table>

The energy consumption model adopted in our work is similar to the model used in [10], where the transmission power dissipation is 0.5 watt and the receiving power
dissipation is 0.2 watt for the wireless network interface. In order to focus on the energy consumption of data transmission and receiving, we do not consider the energy consumed by other components. In order to better understand the overall performance of the system, we introduce a new metric called energy-stretch, which is defined as

\[
\text{energy-stretch} = \text{stretch} \times \text{energy-consumed}
\]

The energy-stretch value gives a better indication of the system performance because it considers two important parameters: energy consumption and stretch. The simulation results consist of two parts: one to evaluate the performance of the VP scheme and the other to evaluate the performance of the AVP schemes.

### 2.4.2 Simulation Results: VP

#### 2.4.2.1 The Effects of the Mean Update Arrival Time

Figure 2.3 shows the effects of the mean update arrival time on the energy consumption, the average stretch, and the energy-stretch. The figure shows various VP approaches with different \( N_p \) value. As we know, the number of prefetches increases as \( N_p \) increases. As a result, the energy consumption increases and the average stretch drops. This has been verified by the simulation results in Figure 2.3(a) and (b). The \( N_p = 10 \) approach has the lowest power consumption, but it has the highest stretch. Figure 2.3(c) shows their energy-stretch value. Although \( VP(N_p = 10) \) has the lowest energy-stretch when \( T_{update} = 10s \), it has higher energy-stretch than \( VP(N_p = 50) \) when \( T_{update} > 40s \), and it has a very high average stretch. Generally speaking, \( VP(N_p = 50) \) has low stretch while consuming similar power as \( VP(N_p = 10) \) when \( T_{update} > 40s \).
(a) Energy consumption

(b) Average stretch

(c) The energy-stretch

Fig. 2.3. Performance under varying update arrival time
We use $N_p = 50$ in the following experiments when VP scheme is used unless specified otherwise.

Compared to the UIR approach, the VP approach saves a large amount of power since the UIR approach aggressively prefetches data to fill up the client cache. From Figure 2.3(a), we can also see that four approaches have similar power consumption when $T_{update} = 10000s$, because only limited number of data items in the cache are invalidated when $T_{update}$ is very large and hence the clients only need to prefetch a few data items. As the mean update arrival time drops, many data items are updated, and the number of prefetches increases. As a result, the power consumption increases. However, the increasing trend is different. The UIR approach has the largest power consumption increase whereas $VP(N_p = 10)$ has the lowest. This is due to the fact that $VP(N_p = 10)$ limits the number of prefetches to 10.

From Figure 2.3(b), we can see that the UIR approach does not follow the same trend as other VP approaches. When the mean update arrival time is low, UIR has lower stretch than $VP(N_p = 10)$ and $VP(N_p = 50)$ since UIR allows aggressive prefetching, while VP only prefetches $N_P$ items with high access rate and low update rate (based on the value function). When the update arrival time is low, data items are frequently invalidated. Due to aggressive prefetch, UIR achieves better performance at the cost of high energy consumption as shown in Figure 2.3(a). As the mean update time increases, various VP approaches outperform UIR, since VP only prefetches the data with high value. Comparing $VP(N_p = 200)$ with UIR, we can see the advantage of VP. Although $VP(N_p = 200)$ only prefetch 200 data items, which is much less that of the UIR approach, it still outperforms UIR in terms of energy, stretch, and energy-stretch.
2.4.2.2 Comparisons with Simple Prefetch Schemes

A prefetch scheme may utilize the fact that the Zipf-like distribution favors items with small id. One simple and reasonable prefetch scheme is to prefetch those k hottest (frequently accessed) data items. In this subsection, we evaluate this simple prefetch scheme with VP. In Figure 2.4, SIMP-k represents the prefetch scheme which prefetches k hottest data items. As expected, smaller k results in lower energy consumption and higher average stretch. Overall, the VP scheme has the best performance considering both energy consumption and average stretch. Although the simple prefetch scheme considers the access frequency factor, it does not consider other factors such as data size, cache invalidation cost. Thus, it underperforms the VP scheme.

From Figure 2.4, we can also see that the energy consumption of the UIR approach and the VP approach both increase with the mean query generate time. This can be explained as follows. When the mean query generate time increases, if other parameters do not change, the update rate within the time of two query interval increases. In other words, increasing the mean query generate time without changing the update arrival time should have the same effect of reducing the update arrival time without changing the mean query generate time. As explained in Section 2.4.2.1, the energy consumption increases if the update arrival time drops, and then the energy consumption increases as the mean query generate time increases.

2.4.2.3 The Effects of the Cache Size

Figure 2.5 shows the effects of the cache size on the average stretch and the energy consumption of the UIR approach and the VP scheme. As can be seen, UIR and VP have
Fig. 2.4. Comparisons with simple prefetch schemes
Fig. 2.5. Performance as a function of the cache size
similar energy consumption when the cache size is small since the number of items to be prefetched is limited by the cache size in the UIR approach. As the cache size increases, the number of prefetches in the UIR approach increases and its energy consumption also increases. This is different from VP. Since $N_p$ is fixed, the data to be prefetched is also fixed in VP. Moreover, when the cache size increases, the cache hit ratio increases, and then more queries can be served from the cache without sending requests and receiving data. Thus, the amount of energy consumed per query actually drops. This explains why the energy consumption per query in VP drops as the cache size increases. As shown in Figure 2.5 (b), due to aggressive prefetching, the average stretch of the UIR approach drops below that of the VP approach when the cache size increases to 25%. When considering both power consumption and stretch, VP always outperforms UIR as shown in Figure 2.5(c).

2.4.2.4 The Effects of the Zipf Parameter $\theta$

The Zipf parameter $\theta$ determines the “skewness” of the access distribution. Figure 2.6 shows the effects of the access pattern on the system performance. When $\theta = 0$, the access pattern follows uniform distribution. That is, all the data items have the same access probability. At this time, prefetch is less effective and the energy-stretch value is high. As $\theta$ increases, more data accesses are focused on items with small id and the energy-stretch drops. Figure 2.6 demonstrates that our approach always outperforms the UIR approach. Even when the access pattern follows uniform distribution ($\theta = 0$), our approach still performs better. It also shows that our approach works for a wide spectrum of access patterns.
Fig. 2.6. The effects of the Zipf parameter $\theta$
2.4.2.5 The Effects of the Access Pattern Variation

In practice, the client access pattern may change. To model such environments, we make the following modifications. The data access distribution is shifted one item every $\delta$ IR intervals. More specifically, suppose item $i$ is the data to be accessed according to the Zipf distribution. An offset $\Delta$ is added to $i$ after each $\delta$ IR intervals so that $i + \Delta$ is the actual data item that will be accessed. $\Delta$ is initialized to 0, and increased by one every $\delta$ IR intervals. The smaller $\delta$ is, the faster the access pattern changes. To make it reasonable, we also apply the same offset to the update so that the items with high update rate still have high access rate.

Figure 2.7 shows the results when the access pattern changes. If the access pattern changes rapidly, the average energy consumption is very high because clients need to spend more energy to prefetch and request data from the server. It can be seen that the VP approach always outperforms the UIR approach.

2.4.3 Simulation Results: Adaptation

2.4.3.1 Adapting $N_p$ to Reach a Target Battery Life

Figure 2.8 evaluates the performance of the AVP$_T$ scheme after the clients are assigned 5000 joule initially. By simulation, we found that the battery life is 190k seconds when $VP(N_p = 0)$ is used. Although the cache size is 20% of the database size, considering the data distribution pattern and the data size difference, the client cache can hold as many as 840 data items. We set $N_p^{max}$ to be 840, but we should know that clients will not prefetch data items with negative value. Based on simulation results, the
Fig. 2.7. The effects of the access pattern variation
Fig. 2.8. AVP\_T: adapting $N_p$ to reach a target battery life
battery life is 110k seconds when $VP(N_p = 840)$ is used. To evaluate the performance of the AVP $T$ scheme, we consider the following target battery life ($T$):

- $T = 100k$ seconds. In this case, denoted as AVP $T$(100, 0), a very short battery life time is expected, and $N_p$ has an initial value of 0.

- $T = 200k$ seconds. In this case, denoted as AVP $T$(200, 840), the battery life is too long to be reachable, and $N_p$ has an initial value of $N_p^{max}$.

- $T = 140k$ seconds. There are two cases: $N_p$ has an initial value of 0, denoted as AVP $T$(140, 0); $N_p$ has an initial value of $N_p^{max}$, denoted as AVP $T$(140, 840).

As shown in the Figure 2.8(a), the adaptive schemes can quickly adjust $N_p$ to reach the target battery life if it is possible. For AVP $T$(200, 840) and AVP $T$(100, 0), because the target battery life is not reachable, these two schemes can only adjust $N_p$ to reach the target as close as possible. The small difference of the battery life time between AVP $T$(200, 840) and $VP(N_p = 0)$ is caused by the aggressive initial prefetch of the AVP $T$(200, 840) scheme. For the two adaptive schemes with moderate target life, AVP $T$(140, 0) and AVP $T$(140, 840), both reaches the target battery life time. The difference between the actual battery life and the target battery life is less than 3%. Figure 2.8(c) shows the cumulative average stretch from the time to collect results. Figure 2.8(d) shows the average stretch of every 2000 seconds, which is the time for $N_p$ to be re-evaluated. As can be seen from 2.8(c), the difference between the final average stretch of AVP $T$(140, 0) and AVP $T$(140, 840) is less than 4%. Figure 2.8(d) also shows that AVP $T$(140, 0) and AVP $T$(140, 840) have similar stretch after the first 60k seconds.
2.4.3.2 The Adaptation of $N_p$ based on the Power Level

In AVP$_P$, when the energy level becomes less than a threshold, the prefetch rate is reduced. When the energy level becomes critically low, the prefetch rate is further reduced. By reducing the prefetch rate, energy can be saved and the system can last longer. Figure 2.9 compares the energy level and the system performance of three schemes: UIR, $VP(N_p = 400)$, and AVP$. In AVP, the initial value of $N_p$ is also 400. As can be seen, the UIR approach has higher prefetch rate and consumes more energy, whereas our approaches have lower energy consumption. Due to energy saving, our adaptive approach can last more than 20% longer than the UIR approach.

$VP(N_p = 400)$ and AVP$_P$ have the same amount of energy consumption when there are abundant energy. As the energy level falls below 50%, AVP$_P$ reduces the prefetch rate to save energy. Figure 2.9 (a) shows that AVP$_P$ uses less energy because of the reduction of $N_p$. As shown in the figure, without adaptation, VP continues to use more energy and eventually runs out of power sooner than AVP$_P$. On the other hand, the power saving of AVP$_P$ is at the cost of increasing the average stretch, although the increase is not that significant. As shown in Figure 2.9(b), the average stretch of AVP$_P$ is only slightly higher than that of VP. Figure 2.9(c) shows that AVP$_P$ manages to achieve low energy-stretch while increasing the battery life.

2.5 Related Work

Prefetching has been widely used to reduce the response time in the Web environment [16, 18, 32, 44]. Most of these techniques concentrate on estimating the probability
Fig. 2.9. The energy level and system performance as a function of time
of each file being accessed in the near future. Since these techniques are designed for the
point-to-point communication environment, they are not suitable for the broadcasting
environment in mobile computing systems.

Recently, prefetching has been used in many cache management schemes for mo-
bile environments [10, 22, 24] to reduce the query latency and the bandwidth consump-
tion. In Grassi’s scheme [24], a push-based model is used. The system performance is
optimized using indexing techniques to periodically re-broadcast the hot data. In their
prefetch scheme, each data item is assigned a calculated value \( f(v(i)) \), which is a function
of the access rate of item \( i \). Based on this value, the client decides whether to prefetch
the item or not when it appears in the broadcast channel. If there exists an item \( j \) in the
client’s cache such that \( f(v(i)) > f(v(j)) \), item \( j \) is removed from the cache and replaced by
item \( i \). This prefetch scheme fails to address a number of issues such as data size, data
update rate, and power consumption.

In [10], Cao proposed an adaptive prefetch scheme. In this scheme, clients record
the number of times a cached item being accessed and prefetched, respectively. The
client calculates the prefetch access ratio (PAR), which is the number of prefetches
divided by the number of accesses, for each item. If PAR is less than one, prefetching
the data is useful since the prefetched data may be accessed multiple times. When power
consumption becomes an issue, the client marks those cache items with \( PAR > \beta \) as
non-prefetch, where \( \beta > 1 \) is a system tuning factor, and should be dynamically changed
based on the energy consumption. However, no clear methodology as to how and when
\( \beta \) should be changed. This scheme, like the previous one [24], does not consider varying
data size and the data update rate.
Gitzenis and Bambos [22] proposed a prefetch scheme considering the quality of the wireless channel. Clients prefetch aggressively when the channel quality is good but reduce the prefetch rate when the channel quality becomes poor. Their scheme assumes that the cost of accessing a data item is already given. Our value-base prefetch scheme actually gives a function to identify the value of each data item and prefetch those items that can minimize the overall access cost. Thus their work complements our work.

2.6 Summary of Prefetch Schemes

Prefetching is an effective technique to reduce the query latency. However, prefetching consumes power. In wireless networks where power is limited, it is essential to correctly identify the data to be prefetched in order to provide better performance and reduce the power consumption. In this chapter, a value-based prefetch scheme was proposed. The proposed scheme evaluates the cost of prefetching a data item by taking into account various factors such as the data size, the access rate, the update rate, and the cache validation delay. In addition to making smarter prefetch decisions, the scheme is designed to be adaptive, adjusting the prefetch rate based on the current energy level. Simulation results verified that the proposed schemes can reduce the energy consumption and improve the system performance in terms of stretch compared to the UIR approach under various system settings.
Chapter 3

Data Access in Multi-hop based Mobile Environments:
Cooperative Caching

3.1 Introduction

Portable computers and wireless networks are becoming widely available, which enable users to remain connected to the Internet while moving around. For example, with cellular network techniques such as General Packet Radio Service (GPRS), mobile users can connect to the Internet while moving. However, mobile users may want to communicate with each other in situations where such kind of fixed infrastructure is not available. For example, a group of emergency rescue workers may need to form a network after an earthquake, or a group of soldiers may need to communicate during a military operation. In such circumstances, a collection of mobile nodes with wireless network interfaces may form a temporary network without the aid of any established infrastructure or centralized administration. This type of network is known as wireless ad hoc network [34].

Wireless ad hoc networks have received considerable attention due to the potential applications in the battlefield, disaster recovery, and outdoor assemblies. Ad hoc networks are ideal in situations where installing an infrastructure is not possible because the infrastructure is too expensive or too vulnerable. Due to lack of infrastructure support, each mobile node in the network acts as a router, forwarding data packets for other
mobile nodes through multi-hop links. Because of this, we use multi-hop based wireless mobile networks and wireless ad hoc networks interchangeably in this thesis. Most of the previous researches [19, 34, 37, 66] in ad hoc networks focus on the development of dynamic routing protocols that can efficiently find routes between two communicating nodes. Although routing is an important issue in ad hoc networks, other issues such as information (data) access are also very important since the ultimate goal of using ad hoc networks is to provide information access to mobile nodes. We use the following two examples to motivate our research on data access in ad hoc networks.

**Example 1:** In a battlefield, an ad hoc network may consist of several commanding officers and a group of soldiers around the officers. Each officer has a relatively powerful data center, and the soldiers need to access the data centers to get various data such as the detailed geographic information, enemy information, and new commands. The neighboring soldiers tend to have similar missions and thus share common interests. If one soldier accessed a data item from the data center, it is quite possible that nearby soldiers access the same data some time later. It saves a large amount of battery power, bandwidth, and time if later accesses to the same data are served by the nearby soldier who has the data instead of the faraway data center.

**Example 2:** Recently, many mobile infostation systems have been deployed to provide information for mobile users. For example, infostations deployed by tourist information center may provide maps, pictures, history of attractive sites. Infostation deployed by a restaurant may provide menus. Due to limited radio range, an infostation can only cover a limited geographical area. If a mobile user, say Jane, moves out of the infostation range, she will not be able to access the data provided by the infostation. However, if
mobile users are able to form an ad hoc network, they can still access the information. In such an environment, when Jane’s request is forwarded to the infostation by other mobile users, it is very likely that one of the mobile nodes along the path has already cached the requested data. Then, this mobile node can send the data back to Jane to save time and bandwidth.

From these examples, we can see that if mobile nodes are able to work as request-forwarding routers, bandwidth and power can be saved, and delay can be reduced. In fact, the cooperative caching technique has been widely used to improve the Web performance [17, 20, 53, 54, 61, 65]. Although cooperative caching and proxy techniques have been extensively studied in wired networks, little has been done to apply this technique to ad hoc networks. Due to mobility and resource constraints, techniques designed for wired networks may not be applicable to ad hoc networks. For example, most researches on cooperative caching in the Web environment assume a fixed topology, but this may not be the case in ad hoc networks due to mobility. Since the cost of the wireless link is different from the wired link, the decision regarding where to cache the data and how to get the cached data may be different.

In this chapter, we design and evaluate cooperative caching techniques to efficiently support data access in ad hoc networks. Specifically, we propose three schemes: CachePath, CacheData and HybridCache. In CacheData, intermediate nodes cache the data to serve future requests instead of fetching data from the data center. In CachePath, mobile nodes cache the data path and use it to redirect future requests to the nearby mobile node which has the data instead of the faraway data center. To further improve the performance, we design a hybrid approach (HybridCache), which can further improve
the performance by taking advantage of CacheData and CachePath while avoiding their weaknesses. To deal with the limited cache size of mobile nodes, a cache replacement policy is also proposed to find the right candidate to be replaced from the cache when new data arrive. Simulation results show that the proposed schemes can significantly improve the performance in terms of the query delay and the message complexity when compared to other caching schemes.

The rest of this chapter is organized as follows. In Section 3.2, we present the CacheData scheme and the CachePath scheme. Section 3.3 presents the HybridCache scheme. The performance of the proposed schemes is evaluated in Section 3.4. Section 3.5 discusses the related work. Section 3.6 summarizes our work on cooperative cache schemes in multi-hop based mobile environments.

3.2 Proposed Basic Cooperative Cache Schemes

In this section, we propose two basic cooperative cache schemes and analyze their performance.

3.2.1 System Model

Figure 3.1 shows part of an ad hoc network. Some nodes in the ad hoc network may have wireless interfaces to connect to the wireless infrastructure such as wireless LAN or cellular networks. Suppose node $N_{11}$ is a data source (center), which contains a database of $n$ items $d_1, d_2, ..., d_n$. Note that $N_{11}$ may be a node connecting to the wired network which has the database.
In ad hoc networks, a data request is forwarded hop-by-hop until it reaches the data center and then the data center sends the requested data back. Various routing algorithms have been designed to route messages in ad hoc networks. To reduce the bandwidth consumption and the query delay, the number of hops between the data center and the requester should be as small as possible. Although routing protocols can be used to achieve this goal, there is a limitation on how much they can achieve. In the following, we propose two basic cooperative caching schemes: \textit{CacheData} and \textit{CachePath}.

3.2.2 Cache the Data (CacheData)

In CacheData, the mobile node caches a passing-by data item $d_i$ locally when it finds that $d_i$ is popular, i.e., there were many requests for $d_i$, or it has enough free cache space. For example, in Figure 3.1, both $N_6$ and $N_7$ request $d_i$ through $N_5$, $N_5$ knows that $d_i$ is popular and caches it locally. Future requests by $N_3$, $N_4$, or $N_5$ can
be served by $N_5$ directly. Since CacheData needs extra space to save the data, it should be used prudently. Suppose the data center receives several requests for $d_i$ forwarded by $N_3$. Nodes along the path $N_3 - N_4 - N_5$ may all think that $d_i$ is a popular item and should be cached. However, it wastes a large amount of cache space if three of them all cache $d_i$. To avoid this, a conservative rule should be followed: a node does not cache the data if all requests for the data are from the same node. As in the previous example, all requests received by $N_5$ are from $N_4$, which in turn are from $N_3$. With the new rule, $N_4$ and $N_5$ do not cache $d_i$. If the requests received by $N_3$ are from different nodes such as $N_1$ and $N_2$, $N_3$ will cache the data. If the requests all come from $N_1$, $N_3$ will not cache the data, but $N_1$ will cache it. Certainly, if $N_5$ receives requests for $d_i$ from $N_6$ and $N_7$ later, it may also cache $d_i$. Note that $d_i$ is at least cached at the requesting node, which can use it to serve the next query.

This conservative rule is designed to reduce the cache space requirement. In some situations, e.g., when the cache size is very large or for some particular data that are interested by most nodes, the conservative rule may decrease the cache performance because data are not cached at every intermediate nodes. However, in mobile networks, nodes usually have limited cache spaces. And we do not assume that some data are interested by all mobile nodes. Therefore, the conservative rule is adopted in our work.

### 3.2.3 Cache the Data Path (CachePath)

The idea of CachePath can be explained by using Figure 3.1. Suppose node $N_1$ has requested a data item $d_i$ from $N_{11}$. When $N_3$ forwards the data $d_i$ back to $N_1$, $N_3$ knows that $N_1$ has a copy of $d_i$. Later, if $N_2$ requests $d_i$, $N_3$ knows that the data center
$N_{11}$ is three hops away whereas $N_1$ is only one hop away. Thus, $N_3$ forwards the request to $N_1$ instead of $N_4$. Note that many routing algorithms (such as AODV [48] and DSR [34]) provide the hop count information between the source and destination. By caching the data path for each data item, bandwidth and the query delay can be reduced since the data can be obtained through less number of hops. However, recording the map between data items and caching nodes increases routing overhead. In the following, we propose some optimization techniques.

When saving the path information, a mobile node need not save all the node information along the path. Instead, it can save only the destination node information, as the path from current router to the destination can be found by the underlying routing algorithm.

In CachePath, a mobile node does not need to record the path information of all passing-by data. For example, when $d_i$ flows from $N_{11}$ to destination node $N_1$ along the path $N_5 - N_4 - N_3$, $N_4$ and $N_5$ need not cache the path information of $d_i$ since $N_4$ and $N_5$ are closer to the data center than the caching node $N_1$. Thus, a node only needs to record the data path when it is closer to the caching node than the data center.

Due to mobility, the node which caches the data may move. The cached data may be replaced due to the cache size limitation. As a result, the node which modified the route should reroute the request to the original data center after it finds out the problem. Thus, the cached path may not be reliable and using it may adversely increase the overhead. To deal with this issue, a node $N_i$ caches the data path only when the caching node, say $N_j$, is very close. The closeness can be defined as a function of its distance to the data center, its distance to the caching node, the route stability, and
the data update rate. Intuitively, if the network topology is relatively stable, the data update rate is low, and its distance to the caching node (denoted as $H(i,j)$) is much lower than its distance to the data center (denoted as $H(i,C)$), the routing node should cache the data path. Note that $H(i,j)$ is a very important factor. If $H(i,j)$ is small, even if the cached path is broken or the data are unavailable at the caching node, the problem can be quickly detected to reduce the overhead. Certainly, $H(i,j)$ should be smaller than $H(i,C)$. The number of hops that a cached path can save is denoted as

$$H_{save} = H(i,C) - H(i,j)$$

where $H_{save}$ should be greater than a system tuning threshold, called $T_H$, when CachePath is used.

**Maintain cache consistency:** There is a cache consistency issue in both CacheData and CachePath. We have done some work [10, 11] on maintaining strong cache consistency in single-hop based wireless environment. However, due to bandwidth and power constraints in ad hoc networks, it is too expensive to maintain strong cache consistency, and the weak consistency model is more attractive. A simple weak consistency model can be based on the Time-To-Live (TTL) mechanism, in which a node considers a cached copy up-to-date if its TTL has not expired, and removes the map from its routing table (or removes the cached data) if the TTL expires. As a result, future requests for this data will be forwarded to the data center.

Due to TTL expiration, some cached data may be invalidated. Usually, invalid data are removed from the cache. Sometimes, invalid data may be useful. As these data
have been cached by the node, it indicates that the node is interested in these data. When a node is forwarding a data item and it finds there is an invalid copy of that data in the cache, it caches the data for future use. To save space, when a cached data item expires, it is removed from the cache while its id is kept in “invalid” state as an indication of the node’s interest. Certainly, the interest of the node may change, and the expired data should not be kept in the cache forever. In our design, if an expired data item has not been refreshed for the duration of its original TTL time (set by the data center), it is removed from the cache.

When cooperative caching is used, mobile nodes need to check passing-by data besides routing. This may involve cross-layer optimization, and it may increase the processing overhead. However, the processing delay is still very low compared to the communication delay. Since most ad hoc networks are specific to some applications, cross-layer optimization can also reduce some of the processing overhead. Considering the performance improvement, the use of cooperative cache is well justified.

3.2.4 Performance Analysis

In this section, we analyze the performance of the proposed schemes. The performance metric is the distance (hop count) between a requester and a node that has the requested data. This node can be a caching node, or the data center when no caching node is found. Reducing the hop count can reduce the query delay, the bandwidth and the power consumption since fewer nodes are involved in the query process. Further, reducing the hop count can also reduce the workload of the data center since requests served by caches will not be handled by the data center.
The notations used in the analysis are as follows:

- $\bar{H}$: the average number of hops between a mobile node and the data center.
- $P_{dd}$: the probability that a data item is in the cache in the CacheData scheme.
- $P_{dp}$: the probability that a data item is in the cache in the CachePath scheme.
- $P_{pp}$: the probability that a path is in the cache in the CachePath scheme.
- $P_i$: the probability that a cached item is not usable. This may be caused by TTL expiration or broken paths because of node movement.
- $L_d$: in CacheData, the average length of the path for a request to reach the node (or the original server) which has a valid copy of the data. If the requester has a valid copy of the data, $L_d = 1$ for easy of presentation.
- $L_p$: in CachePath, the average length of the path for a request to reach the node (or the original server) which has a valid copy of data. $L_p = 1$ if the requester has a valid copy of the data.

We make some assumptions to simplify the analysis. For example, we assume that parameters such as $P_{dd}$, $P_{dp}$, $P_{dp}$, and $P_i$ in all nodes are the same. These assumptions allow us to study the ad hoc network where these parameters are affected by various factors such as cache size, topology changes, node/link failures, etc. Most of these factors are not easy to model, especially when they can affect each other. The simulation results in Section 3.4 match the analytical results and verify that these assumptions are reasonable.
Given these notations, we can obtain the expected number of hops that a request takes from node $N_i$ to the node which has the data. Let $P'_d = P_{dd}(1 - P'_i)$, then

$$L_d = P'_d \cdot 1 + (1 - P'_d) \cdot P'_d \cdot 2 + \ldots + (1 - P'_d) \cdot H(i, C) \cdot P'_d \cdot H(i, C)$$

$$= \sum_{k=1}^{H(i, C)} (1 - P'_d)^{k-1} \cdot P'_d \cdot k$$

$$\approx \frac{1}{P'_d} = \frac{1}{P_{dd}(1 - P'_i)}$$

This equation is an approximation of $L_d$ since in practice $P_{dd}$ may be different at different nodes. Equation (3.1) helps us understand the effects of many important factors, and we believe the approximation is reasonable. Note that $L_d$ is bounded by $\overline{H}$. When $P'_d$ is not too small, i.e., not less than $1/\overline{H}$, line 4 of Equation (3.1) provides an adequate approximation.

To calculate $L_p$, three cases need to be considered:

1. The requested data item is in the local cache.

2. A path is found in the local cache which indicates $N_i$ caches the requested data. Two sub-cases are possible:
   
   (a) a valid data item is found in $N_i$.
   
   (b) the data item in $N_i$ is not usable because of broken path or TTL expiration.

3. No data or path is found in the local cache.

Let $P'_p = P_{dp}(1 - P'_i)$. The probabilities of Cases 1, 2(a), 2(b), and 3 are $P'_p$, $(1 - P'_p)P_{pp}(1 - P'_i)$, $(1 - P'_p)P_{pp}P'_i$, and $(1 - P'_p)(1 - P_{pp})$ respectively. The number of hops needed for a request to get the data is 1 for Case 1 and $1 + L_p$ for Case 2(a) and Case 3. Note that for Case 3, the distance is not $\overline{H}$ because intermediate nodes
also check their local cache for the requested data or path. Thus, it is different from forwarding the request directly to the data center. For Case 2(b), the request need to travel $1 + L_p$ to reach $N_i$. Then it is redirected to the data center which is $H$ away. At last, the data item is sent back to the requester in $H$ hops. Therefore, the average number of hops needed for the request\(^1\) is $(1 + L_p + H + H)/2 = H + (1 + L_p)/2.$

Thus

$$L_p = P_p' \cdot 1 + (1 - P_p') \cdot P_{pp}$$

$$= (P_i(H + \frac{L_p + 1}{2}) + (1 - P_i)(L_p + 1))$$

$$+ (1 - P_p')(1 - P_{pp})(1 + L_p)$$

(3.2)

So,

$$L_p = \frac{P_p' + (1 - P_p')P_{pp}((P_iH - P_{pp} + 1)) + (1 - P_p')(1 - P_{pp})}{1 - (1 - P_p')P_{pp}(1 - \frac{P_p}{2}) - (1 - P_p')(1 - P_{pp})}$$

(3.3)

In Equation (3.3), $P_{pp}$ is specific to CachePath. Therefore, it needs to be fixed when comparing $L_p$ to $L_d$. If $P_{pp} = 0$, $L_p = 1/(P_{dp}(1 - P_i))$ and if $P_{pp} = 1$,

$$L_p = \frac{P_{dp}(1 - P_i)(\frac{P_i}{2} - P_iH) + (P_iH - \frac{P_i}{2} + 1)}{1 - (1 - P_{dp}(1 - P_i))(1 - \frac{P_i}{2})}$$

(3.4)

$P_{pp} = 1$ gives the performance upper bound of CachePath. Equations (3.1) and (3.4) are still complex as they contain several parameters. We can fix some parameters to get a better understanding of the relation between $L_d$ and $L_p$.

\(^{1}\)The average number of hops is the round-trip distance divided by two.
Suppose $P_t = 0$ (i.e., all the data items in the cache are valid), we have

\[ L_d = \frac{1}{P_{dd}} \quad \text{and} \quad L_p = \frac{1}{P_{dp}} \quad (3.5) \]

CachePath needs less cache space to store extra data\(^2\). Therefore $P_{dd} < P_{dp}$ when the cache size is not very big, which means $L_p < L_d$.

![Average path length difference](image.png)

Fig. 3.2. Performance comparison

Suppose $P_{dd} = P_{dp}$. This assumption favors CacheData since in practice, the cache size is limited and $P_{dd} < P_{dp}$. Figure 3.2 shows some numerical results of CachePath and CacheData by comparing their average path lengths $L_p$ and $L_d$ under two

\(^2\)Note that a cached path only contains the final destination node id, as explained in Section 3.2.3. We assume that the size of any data item is larger than the size of a data id.
different $\overline{H}$ values: five and ten. As can be seen, in both cases, $L_p$ is similar to $L_d$ when $P_i$ is small, which shows the advantage of CachePath considering that the assumption $P_{dd} = P_{dp}$ favors CacheData when the cache size is small. If the cache size is large enough so that $P_{dp}$ is similar to $P_{dd}$, CacheData performs better as $L_d$ is similar to or less than $L_p$. When $P_i$ is high, the difference between $L_p$ and $L_d$ is also high, because many requests follow paths that lead to data not useful in CachePath when $P_i$ is high, which is essentially *chasing the wrong path*. In such situation, it is better to adopt CacheData because it does not redirect requests.

Comparing these two cases: $\overline{H} = 5$ and $\overline{H} = 10$ in Figure 3.2, we can see that large $\overline{H}$ value decreases the performance of CachePath when compared to CacheData. This is because the penalty of chasing a wrong path is high when the network size is large (large $\overline{H}$). We can summarize the analytical results as follows:

- Both schemes can reduce the average number of hops between the requester and the node which has the requested data. For example, when $P_i = 0$, the number of hops can be reduced if the cache hit ratio is greater than $1/\overline{H}$. If there is no cached data or path available, our schemes fall back to traditional caching scheme, where requests are sent directly to the data center.

- When the cache size is small, CachePath is better than CacheData; when the cache size is large, CacheData is better.

- When the network size is small (small $\overline{H}$), CachePath is a good approach; when the network size is large, CacheData performs better.
• When the data items are updated slowly or mobile nodes move slowly, i.e., $P_i$ is small, CachePath is a good approach; in other cases, CacheData performs better.

3.3 A Hybrid Caching Scheme (HybridCache)

The performance analysis showed that CachePath and CacheData can significantly improve the system performance. We also found that CachePath performs better in some situations such as small cache size or low data update rate, while CacheData performs better in other situations. To further improve the performance, we propose a hybrid scheme HybridCache to take advantage of CacheData and CachePath while avoiding their weaknesses. Specifically, when a node forwards a data item, it caches the data or path based on some criteria. These criteria include the data item size $s_i$, the TTL time $TTL_i$, and the $H_{save}$. For a data item $d_i$, the following heuristics are used to decide whether to cache data or path:

• If $s_i$ is small, CacheData should be adopted because the data item only needs a very small part of the cache; otherwise, CachePath should be adopted to save cache space. The threshold value for data size is denoted as $T_s$.

• If $TTL_i$ is small, CachePath is not a good choice because the data item may be invalid soon. Using CachePath may result in chasing the wrong path and end up with re-sending the query to the data center. Thus, CacheData should be used in this situation. If $TTL_i$ is large, CachePath should be adopted. The threshold value for $TTL$ is a system tuning parameter and denoted as $T_{TTL}$. 
• If $H_{safe}$ is large, CachePath is a good choice because it can save a large number of hops; otherwise, CacheData should be adopted to improve the performance if there is enough empty space in the cache. We adopt the threshold value $T_H$ used in CachePath as the threshold value.

These thresholds values should be set carefully as they may affect the system performance. Their effects and how to set them are studied through simulations in Section 3.4.2.1 and 3.4.2.2.

Figure 3.3 shows the algorithm that applies these heuristics in HybridCache. In our design, caching a data path only needs to save a node id in the cache. This overhead is very small. Therefore, in HybridCache, when a data item $d_i$ needs to be cached using CacheData, the path for $d_i$ is also cached. Later, if the cache replacement algorithm decides to remove $d_i$, it removes the cached data while keeping the path for $d_i$. From some point of view, CacheData degrades to CachePath for $d_i$. Similarly, CachePath can be upgraded to CacheData again when $d_i$ passes by.

3.3.1 Cache Replacement Policy

Because of limited cache size, cache replacement policy must be adopted to evict data from the cache when new data arrive. One widely used cache replacement policy is LRU, which removes the least-recently-used data from the cache. However, some researches ([56, 70]) show that LRU can be outperformed by policies that consider various system parameters such as the data size, transfer time, data invalidation rate, etc. The problem with policies proposed in [56, 70] is that they require the input of many system parameters that are constantly changing and not easy to estimate.
(A) When a data item $d_i$ arrives:

\[
\begin{align*}
\text{if } (d_i \text{ is the requested data by the current node}) & \text{ then } \\
& \phantom{=} \text{cache data item } d_i; \text{ return;}
\end{align*}
\]

/* Data passing by */

\[
\begin{align*}
\text{if } (\text{an old version of } d_i \text{ is in the cache}) & \text{ then } \\
& \phantom{=} \text{update the cached copy;}
\end{align*}
\]

\[
\begin{align*}
\text{else if } & (s_i < T_s \text{ or there is an invalid copy in the cache} \\
& \text{or there is a cached path for } d_i) \text{ then } \\
& \phantom{=} \text{cache data item } d_i;
\end{align*}
\]

\[
\begin{align*}
\text{else if } & (H_{save} > T_H \text{ and } TTL_i > T_{TTL}) \text{ then } \\
& \phantom{=} \text{cache the path of } d_i;
\end{align*}
\]

(B) When cache replacement is necessary:

\[
\begin{align*}
\text{while } & (\text{not enough free space and there are invalid data items in the cache}) \text{ do } \\
& \phantom{=} \text{Remove an invalid data item;}
\end{align*}
\]

\[
\begin{align*}
\text{while } & (\text{not enough free space}) \text{ do } /*\text{still need space}*/*/ \\
& \phantom{=} \text{Remove a valid data item;}
\end{align*}
\]

(C) When a request for data item $d_i$ arrives:

\[
\begin{align*}
\text{if } (\text{there is a valid copy in cache}) & \text{ then } \\
& \phantom{=} \text{send } d_i \text{ to the requester;}
\end{align*}
\]

\[
\begin{align*}
\text{else if } (\text{there is a valid path for } d_i \text{ in the cache}) & \text{ then } \\
& \phantom{=} \text{forward the request to the caching node;}
\end{align*}
\]

\[
\begin{align*}
\text{else } \\
& \phantom{=} \text{forward the request to the data center;}
\end{align*}
\]

Fig. 3.3. The hybrid caching scheme
In our work, we focus on two parameters that are easier to get. The first parameter is the data size $s_i$. Data with larger size are better candidate for replacement because they occupy a large amount of cache space. Replacing them can make room for more incoming data items. The second parameter is $Order(d_i)$, the order of $d_i$ according to the access interest. Let $k = Order(d_i)$, then $d_i$ is the $k^{th}$ most frequently accessed data.

Intuitively, data that are less likely to be accessed should be replaced first. Our policy, called the $Size*Order$ cache replacement policy (SXO), combines these two parameters in the following value function:

$$\text{value}(d_i) = s_i \ast Order(d_i) \quad (3.6)$$

The data item with the largest $\text{value}(d_i)$ is replaced from the cache first.

In Equation (3.6), $s_i$ is known to mobile nodes because $d_i$ is in the cache. Although $Order(d_i)$ is not available, it can be derived from the mobile node’s access rate to $d_i$, denoted as $a_i$. In order to get $a_i$, we can apply similar techniques used by Shim et al. [56] as follows:

$$a_i = \frac{K}{T - T_{a_i}(K)} \quad (3.7)$$

where $T$ is the current time, $T_{a_i}(K)$ is the time of the $K^{th}$ most recent access. If less than $K$ samples are available, all available samples are used. It is shown in [56] that $K$ can be as small as two or three to achieve the best performance. Thus the spatial overhead to store recent access time is relatively small. Once the access frequency to each data item is available, it is easy to get the $Order(d_i)$ value.
Because $Order(d_i)$ is given through estimation, we cannot guarantee that $value(d_i)$ used in the cache replacement is absolutely accurate. Therefore, the SXO policy should remain effective even if $Order(d_i)$ is not very accurate. In our simulations, the sensitivity of SXO to data inaccuracy is studied by introducing noise to $Order(d_i)$. The result shows that SXO is able to perform well even when $Order(d_i)$ is not very accurate.

### 3.4 Performance Evaluation

The performance evaluation includes four sub sections. The simulation model is given in Section 3.4.1. In Section 3.4.2, we verify the analytical results of CacheData and CachePath, and compare them to HybridCache and SimpleCache, which is the traditional cache scheme that only caches the received data at the query node. Section 3.4.3 compares HybridCache to SimpleCache and the cooperative caching scheme proposed by Lau et al. [38], referred to as FloodCache. FloodCache is designed for accessing multimedia data in ad hoc networks. When a query comes, it relies on flooding to find the nearest node that has the requested data. In both Section 3.4.2 and 3.4.3, all schemes use LRU as the cache replacement policy. In Section 3.4.4, we study the effect of cache replacement policies on the query delay, and compare the performance of HybridCache-SXO, which is the HybridCache scheme using the SXO cache replacement policy, to SimpleCache and HybridCache which use LRU.
3.4.1 The Simulation Model

The simulation is based on ns-2 [42] with the CMU wireless extension. In our simulation, both AODV [48] and DSDV [47] were tested as the underlying routing algorithm. Because our schemes do not rely on specific routing protocols, the results from AODV and DSDV are similar. To save space, only the results based on DSDV are shown here.

The node density is changed by choosing the number of nodes between 50 and 100 in a fixed area. We assume that the wireless bandwidth is 2 Mb/s, and the radio range is 250m.

**The node movement model:** We model a group of nodes moving in a 1500m × 320m rectangle area, which is similar to the model used in [66]. The moving pattern follows the random way point movement model [7]. Initially, nodes are placed randomly in the area. Each node selects a random destination and moves toward the destination with a speed selected randomly from (0 m/s, \(v_{\text{max}}\) m/s). After the node reaches its destination, it pauses for a period of time and repeats this movement pattern. Two \(v_{\text{max}}\) values, 2 m/s and 20 m/s, are studied in the simulation.

**The client query model:** The client query model is similar to what have been used in previous studies [10, 70]. Each node generates a single stream of read-only queries. The query generate time follows exponential distribution with mean value \(T_{\text{query}}\). After a query is sent out, the node does not generate new query until the query is served. The access pattern is based on Zipf – like distribution [73], which has been frequently used
[6] to model non-uniform distribution. In the Zipf-like distribution, the access probability of the $i^{th}$ ($1 \leq i \leq n$) data item is represented as follows:

$$P_{a_i} = \frac{1}{i^\theta \sum_{k=1}^{n} \frac{1}{k^\theta}}$$

where $0 \leq \theta \leq 1$. When $\theta = 1$, it follows the strict Zipf distribution. When $\theta = 0$, it follows the uniform distribution. Larger $\theta$ results in more “skewed” access distribution.

The access pattern of mobile nodes can be location-dependent; that is, nodes that are around the same location tend to access similar data, such as local points of interests. To simulate this kind of access pattern, a “biased” Zipf-like access pattern is used in our simulation. In this pattern, the whole simulation area is divided into 10 (X axis) by 2 (Y axis) grids. These grids are named grid 0, 1, 2,... 19 in a column-wise fashion. Clients in the same grid follow the same Zipf pattern, while nodes in different grids have different offset values. For example, if the generated query should access data $id$ according to the original Zipf-like access pattern, then in grid $i$, the new $id$ would be $(id + n \mod i) \mod n$, where $n$ is the database size. This access pattern can make sure that nodes in neighboring grids have similar, although not the same, access pattern.

**The server model:** Two data servers: server0 and server1 are placed at the opposite corners of the rectangle area. There are $n$ data items at the server side and each server maintains half of the data. Data items with even $ids$ are saved at server0 and the rests are at server1. The data size is uniformly distributed between $s_{min}$ and $s_{max}$. The data are updated only by the server. The servers serve the requests on FCFS (first-come-first-service) basis. When the server sends a data item to a mobile node, it sends the TTL
tag along with the data. The TTL value is set exponentially with a mean value. After the TTL expires, the node has to get the new version of the data either from the server or from other nodes before serving the query.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default value</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database size $n$</td>
<td>1000 items</td>
<td></td>
</tr>
<tr>
<td>$s_{\text{min}}$ (KB)</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$s_{\text{max}}$ (KB)</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Number of nodes</td>
<td>100</td>
<td>50 to 100</td>
</tr>
<tr>
<td>$v_{\text{max}}$ (m/s)</td>
<td>2</td>
<td>2 to 20</td>
</tr>
<tr>
<td>Bandwidth (Mb/s)</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>TTL (secs)</td>
<td>5000</td>
<td>200 to 10000</td>
</tr>
<tr>
<td>Pause time (secs)</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>Client cache size (KB)</td>
<td>800</td>
<td>200 to 1200</td>
</tr>
<tr>
<td>Mean query generate time $T_{\text{query}}$ (secs)</td>
<td>5</td>
<td>1 to 100</td>
</tr>
<tr>
<td>Zipf parameter $\theta$</td>
<td>0.8</td>
<td>0 to 1</td>
</tr>
<tr>
<td>$T_H$</td>
<td>2</td>
<td>1 to 5</td>
</tr>
<tr>
<td>$T_s$ (% of $(s_{\text{min}} + s_{\text{max}})$)</td>
<td>40</td>
<td>10 to 100</td>
</tr>
<tr>
<td>$T_{\text{TTL}}$ (secs)</td>
<td>5000</td>
<td>500 to 10000</td>
</tr>
</tbody>
</table>

Most system parameters are listed in Table 3.1. The second column lists the default values of these parameters. In the simulation, we may change the parameters to study their impacts. The ranges of the parameters are listed in the third column. For each workload parameter (e.g., the mean TTL time or the mean query generate time), the mean value of the measured data is obtained by collecting a large number of samples such that the confidence interval is reasonably small. In most cases, the 95% confidence interval for the measured data is less than 10% of the sample mean.
3.4.2 Simulation Results: HybridCache

Experiments were run using different workloads and system settings. The performance analysis presented here is designed to compare the effects of different workload parameters such as cache size, mean query generate time, node density, node mobility, and system parameters such as TTL and $\theta$ on the performance of SimpleCache, CacheData, CachePath, and HybridCache.

3.4.2.1 Fine-tuning CachePath

As stated in Section 3.2.3, the performance of CachePath is affected by the threshold value $T_H$ as a path is only cached when its $H_{save}$ value is greater than $T_H$. A small
$T_H$ means more paths are cached, but caching too many less-valuable paths may increase the delay because the cached paths are not very reliable. A large $T_H$ means only some valuable paths are cached. However, if $T_H$ is too large, many paths are not cached because of the high threshold. As shown in Figure 3.4, $T_H = 2$ achieves a balance, and we use it in the rest of our simulations.

3.4.2.2 Fine-tuning HybridCache

![Graphs showing fine-tuning HybridCache](image)

Fig. 3.5. Fine-tune HybridCache

In HybridCache, if a data item size is smaller than $T_s$, it is cached using CacheData. If $T_s$ is too small, HybridCache fails to identify some small but important data items; if it is too large, HybridCache caches all the data using CacheData. To find an optimal value for $T_s$, we measure the query delay as a function of $T_s$. As $T_s$ is related to
data size, in Figure 3.5 (a), we use a relative value: $T_s/(S_{min} + S_{max})$, which can give us a clearer idea of what the threshold value should be.

As shown in Figure 3.5 (a), when the threshold value increases from 10% to 40%, the query delay drops sharply since more data are cached. If the threshold value keeps increasing beyond 40%, more passing-by data are cached, and the cache has less space to save the accessed data. As a result, some important data may be replaced, and the delay increases. We find that a threshold value of 40% gives the best performance.

Figure 3.5 (b) shows the effect of $T_{TTL}$ on the average query delay. The lowest query delay is achieved when $T_{TTL} = 5000$ seconds. Compared to Figure 3.5 (a), the performance difference between different $T_{TTL}$ is not significant. This is because the database we studied has heterogeneous data size. Data size varies from 1 KB to 10 KB. As data size is a very important factor for caching, it makes the effect of $T_{TTL}$ less obvious.

### 3.4.2.3 Effects of the Cache Size

Figure 3.6 shows the impacts of the cache size on the cache hit ratio and the average query delay. Cache hits can be divided into three categories: *local data hit* which means that the requested data item is found in the local cache, *remote data hit* which means that the requested data item is found in one of the intermediate node when the request is forwarded in the network, and *path hit* which means that a path is found for the request and a valid data item is found in the destination node of that path. Both remote data hit and path hit are considered as remote cache hit because the data are retrieved from remote nodes.
From Figure 3.6 (a), we can see that the local hit ratio of SimpleCache is always the lowest. When the cache size is small, CacheData performs similar to SimpleCache because small cache size limits the aggressive caching of CacheData. When the cache size is large, CacheData can cache more data for other nodes. These data can be used locally and hence the local data hit ratio increases. CachePath does not cache data for other nodes, but its cached data can be refreshed by the data passing by. Therefore, its local data hit ratio is still slightly higher than that of SimpleCache. HybridCache prefers small data items when caching data for other nodes. Therefore, it can accommodate more data and achieve a high local data hit ratio.

Although CacheData and CachePath have similar local data hit ratio in most cases, CacheData always has higher remote data hit ratio because it caches data for other nodes. Especially when the cache size is large, more data can be cached in CacheData and its remote data hit ratio is significantly higher than that of CachePath. HybridCache
has a high remote data hit ratio due to similar reason for its high local data hit ratio. Even if the path hit is not considered, HybridCache still has highest cache hit ratio in most cases. It is worth noticing that CachePath and HybridCache almost reach their best performance when the cache size is 800 KB. This demonstrates their low cache space requirement. This particularly shows the strength of HybridCache as it also provides the best performance at the same time.

Because of the high cache hit ratio, the proposed schemes perform much better than SimpleCache (see Figure 3.6). Comparing CachePath with CacheData, when the cache size is small, CachePath has lower query delay because its path hit helps reduce the average hop count. When the cache size is greater than 800 KB, these two schemes have similar total cache hit ratio, but CacheData has higher local data hit ratio and remote data hit ratio. Because the average hop count of local and remote data hit is lower than that of path hit, CacheData achieves low query delay. This figure also agrees with the performance analysis of CachePath and CacheData in Section 3.2.4.

Comparing all three proposed schemes, we can see that HybridCache performs much better than CacheData or CachePath, because HybridCache applies different schemes (CacheData or CachePath) to different data items, taking advantages of both CacheData and CachePath. As the result of the high local data hit ratio, remote data hit ratio and overall cache hit ratio, HybridCache achieve the best performance compared to other schemes.
Fig. 3.7. The average query delay as a function of the mean query generate time $T_{\text{query}}$

3.4.2.4 Effects of the Query Generate Time

Figure 3.7 shows the average query delay as a function of the $T_{\text{query}}$. Both low mobility ($V_{\text{max}} = 2 \text{ m/s}$) and high mobility ($V_{\text{max}} = 20 \text{ m/s}$) settings are studied. We notice that all the trends are similar except CachePath. There are cases that CachePath even performs worse than SimpleCache. This is due to the fact that high node mobility causes more broken paths, which affects the performance of CachePath. In high mobility setting, CacheData performs better and HybridCache still performs the best in most cases.

When $T_{\text{query}}$ is small, more queries are generated and the system workload is high. As a result, the average query delay is high. As $T_{\text{query}}$ increases, less queries are generated and the average query delay drops. If $T_{\text{query}}$ keeps increasing, the average query delay only drops slowly or even increases slightly. The reason is that the query generating speed is so low that the number of cached data is small and many cached
data are not usable because their TTL have already expired before queries are generated for them. Figure 3.7 verifies this trend.

Under heavy system workload ($T_{query}$ is small), HybridCache can reduce the query delay by as much as 40% compared to CacheData or CachePath. When the system workload is extremely light, the difference between different schemes is not very large. This is because under extreme light workload, the cache hit ratio is low. Therefore, most of the queries are served by the remote data center and different schemes perform similarly.

We can also find that when the query generating speed increases ($T_{query}$ decreases), the delay of HybridCache does not increase as fast as other schemes. This demonstrates that HybridCache is less sensitive to workload increases and it can handle much heavier workload.

3.4.2.5 Effects of the Zipf Parameter $\theta$

The Zipf parameter $\theta$ defines the access pattern of mobile nodes. When $\theta$ is small, the access distribution is more like a uniform distribution. The average query delay is high since the cache is not large enough to save all the data. When $\theta$ is large, the access is focused on the hot (frequently accessed) data, and the average query delay is lower since most of these hot data can be cached. By changing $\theta$, we can see how different access patterns affect the performance. As shown in Figure 3.8, our schemes perform much better than the SimpleCache scheme because the cooperation between nodes can significantly reduce the query delay.
3.4.2.6 Effects of TTL

Figure 3.9 shows the average query delay when the TTL varies from 200 seconds to 10000 seconds. TTL determines the data update rate. Higher update rate (smaller TTL) makes the cached data more likely to be invalidated, and hence the average query delay is higher. When the TTL is very small (200 sec), all four schemes perform similarly, because most data in the cache are invalid and then the cache hit ratio is very low. Since SimpleCache does not allow nodes to cooperate with other nodes, its average query delay does not drop as fast as our schemes when TTL increases. The delay of our schemes drops much faster as TTL increases because nodes cooperate with each other to maximize the benefit of low update rate.

Comparing CachePath to CacheData, CacheData performs better when TTL is small, whereas CachePath performs better when TTL is big. This result again agrees
with the performance analysis. HybridCache further reduces the query delay by up to 45%.

3.4.2.7 Effects of the Node Density

Figure 3.10 shows the average query delay as a function of the number of nodes in the system. As node density increases, the delay of all four schemes increases, because more nodes compete for limited bandwidth. However, the delay of our schemes increases much slower than SimpleCache. This can be explained by the fact that more data can be shared as the number of nodes increases in our schemes, which helps reduce the query delay. When the total number of nodes is small, HybridCache performs similar as CacheData and CachePath. When the number of nodes increases, HybridCache performs much better than other schemes. This indicates that HybridCache scales well with the number of nodes.
3.4.3 Simulation Results: Comparisons

In this section, we compare the performance of the HybridCache scheme to the SimpleCache scheme and the FloodCache scheme in terms of the query delay and the message complexity. A commonly used message complexity metric is the total number of messages injected into the network by the query process [38]. Since each broadcast message is processed (received and then re-broadcasted or dropped) by every node that received it, “the number of messages processed per node” is used as the message complexity metric to reflect the efforts (battery power, CPU time, etc.) of the mobile node to deal with the messages.
3.4.3.1 Effects of the Cache Size

Figure 3.11 shows the impacts of the cache size on the system performance. Figure 3.11 (a) shows that the query delay decreases as the cache size increases. After the cache size increases beyond 800 KB, mobile nodes have enough cache size and the query delay does not drop significantly. The SimpleCache scheme is outperformed by cooperative caching schemes under different cache size settings. This demonstrates that mobile nodes can benefit from sharing data with each other.

FloodCache performs better compared to HybridCache in terms of the query delay. This is because by flooding, FloodCache can find the nearest node that caches the requested data, which reduces the query delay. However, Figure 3.11 (b) shows that HybridCache incurs much less message overhead than FloodCache. The message overhead of HybridCache is even less than that of SimpleCache. The reason is that HybridCache gets data from nearby nodes instead of the faraway data center if possible. Therefore, the data requests and replies need to travel less number of hops and mobile nodes need to process less number of messages. As the cache size increases, the cache hit ratio of HybridCache increases and its message overhead decreases. Because FloodCache uses flooding to find the requested data, it incurs much higher message overhead compared to SimpleCache and HybridCache.

In FloodCache the request is sent out through flooding, and multiple copies of data replies may be returned to the requester by different nodes that have the requested data. In SimpleCache and HybridCache, this can not happen because only one request is sent out for each query in case of local cache miss. Figure 3.11 (c) shows that more than seven
Fig. 3.11. The performance as a function of the cache size
copies of data replies are returned per query in FloodCache. The number of duplicated data replies increases slightly as the cache size increases because data can be cached in more nodes. In our simulation, the data size is relatively small (from 1 KB to 10 KB), and hence the duplicated messages do not affect the performance significantly. For some other environments such as multimedia accessing, transmitting duplicated data messages may waste much more power and bandwidth. As one solution, instead of sending the data to the requester upon receiving a request, mobile nodes which have the data may send back an acknowledgment. The requester can then send another unicast request to the nearest node among them to get the data. The drawback of this approach is that the query delay will be significantly increased.

### 3.4.3.2 Effects of the Mean Query Generate Time $T_{query}$

![Graph](attachment:image.png)

(a) Query delay (b) Message overhead

Fig. 3.12. The performance as a function of the mean query generate time $T_{query}$
Figure 3.12 shows the effects of $T_{query}$ on system performance. HybridCache performs similar to FloodCache when $T_{query}$ is small. When the system workload is low ($T_{query}$ is large), the difference between HybridCache and FloodCache increases. As explained in Section 3.4.2.4, when $T_{query}$ is large, many cached data are not usable because of the TTL expiration. Therefore, the cache hit ratio is very low. Because FloodCache can find the nearest valid data, its query delay is small. HybridCache may not find the valid data item before a request reaches the data center. Therefore, its query delay is a little bit longer. Figure 3.12 (a) shows that the query delay of FloodCache is the lowest. However, as can be seen from Figure 3.12 (b), the message overhead of FloodCache is significantly higher than that of HybridCache.

When considering the results from both Figure 3.11 and Figure 3.12, we can see that FloodCache uses significantly higher message overhead to get a very small query delay improvement over HybridCache. Thus, FloodCache may not be suitable for ad hoc networks where bandwidth and power are scarce. HybridCache performs well because it reduces the query delay compared to SimpleCache and incurs much less overhead compared to FloodCache.

3.4.4 Simulation Results: Cache Replacement Policy

The effect of the cache replacement policy is shown in Figure 3.13. HybridCache-SXO denotes the HybridCache scheme that applies the SXO cache replacement policy. In order to study the effect of inaccurate input on SXO, noise is introduced by changing
the value function to:

\[ \text{value}(d_i) = s_i \ast (\text{Order}(d_i) + \text{uniform}(0, 20)). \]

Note that the \( \text{Order}(d_i) \) used in SXO is already an estimated value. Here, more noise \( (\text{uniform}(0, 20)) \) is added to the estimated value to test the robustness of our scheme. The result is shown in Figure 3.13 as HybridCache-SXO(Err).

Figure 3.13 shows that HybridCache-SXO clearly outperforms HybridCache when the cache size is small, since data size is a very important factor for cache replacement in SXO. Because mobile nodes usually have limited cache size, HybridCache-SXO is suitable for them. Even when noise is deliberately introduced to \( \text{value}(d_i) \), the performance is almost the same as before, which shows the robustness of the proposed cache replacement policy.
When cache size is large enough (more than 800KB), the difference between HybridCache-SXO and HybridCache becomes small. When mobile nodes have enough cache space, there is always enough space for new data, and then cache replacement policies does not affect the cache performance too much. In such cases, we prefer HybridCache due to its low complexity.

3.5 Related Work

3.5.1 Caching Schemes in Wired Networks

Cooperative caching has been widely studied in the Web environment. These protocols can be classified as message-based, directory-based, hash-based, or router-based. Wessels and Claffy introduced the Internet cache protocol (ICP) [61], which has been standardized and is widely used. As a message-based protocol, ICP supports communication between caching proxies using a query-response dialog. It has been reported that ICP scales poorly with increasing number of caches. Directory-based protocols for cooperative caching enable caching proxies to exchange information about cached content. The information is compressed using arrays of bits. Notable protocols of this class include Cache Digests [54] and summary cache [20]. The most notable hash-based cooperative caching protocol constitutes the cache array routing protocol (CARP) [53]. The rational behind CARP constitutes load distribution by hash routing among Web proxy cache arrays. Wu and Yu introduced several improvements to hash-based routing, considering network latency and allowing local replication [65]. Web cache coordination protocol (WCCP) [17], as a router-based protocol, transparently distributes
requests among a cache array. These protocols usually assume fixed network topology and often require high computation and communication overhead. However, in an ad hoc network, the network topology changes frequently. Also, mobile nodes have resource (battery, CPU, and wireless channel) constraints and cannot afford high computation or communication overhead. Therefore, existing techniques designed for wired networks may not be applied directly to ad hoc networks.

### 3.5.2 Caching Schemes in Wireless Networks

Most of the previous research [19, 34, 37, 66] in ad hoc networks focus on routing, and not much work has been done on data access. The directed diffusion proposed by Intanagonwiwat et al. [31] addressed the cooperation among sensor nodes during data collection. Ye et al. [68] applied the query-forwarding concept to sensor networks. They proposed a two-tier data dissemination (TTDD) model for wireless sensor networks. TTDD requires the construction of a grid structure in fixed sensor networks. The nodes at the grid crossing points work as routers, which forward queries to the source and forward data to the sink. Although both approaches use cache, their focus is on data aggregation and compression for sensor networks, not on cooperative caching and data access.

To effectively disseminate data in ad hoc networks, data replication and caching can be used. Data replication schemes in ad hoc networks have been studied by Hara [26]. However, these schemes may not be very effective due to the following reasons: First, because of frequent node movement, powering off or failure, it is hard to find stable nodes to host the replicated data; Second, the cost of initial distribution of the
replicated data and the cost of redistributing the data to deal with node movement or failure is very high.

Similar to the idea of cooperative caching, Papadopouli and Schulzrinne [45] proposed a 7DS architecture, in which a couple of protocols are defined to share and disseminate data among users that experience intermittent connectivity to the Internet. It operates either on a prefetch mode to gather data for serving the users’ future needs or on an on-demand mode to search for data on a single-hop multicast basis. Unlike our work, they focus on data dissemination instead of cache management. Further, their focus is single-hop environment instead of multi-hop environment as in our work.

A cooperative caching scheme designed specifically for accessing multimedia objects in ad hoc networks has been proposed by Lau et al. [38]. When a query comes, this scheme relies on flooding to find the nearest node that have the requested object. Using flooding can reduce the query delay since the request may be served by a nearby node instead of the data center faraway. Thus, it is good for multimedia applications which have strict delay requirements. Another benefit of using flooding is that multiple nodes that contain the requested data can be found. If the data size is very large, when the link to one node fails, the requester can switch to other nodes to get the rest of the requested data. Using flooding incurs significant message overhead. To reduce the overhead, in [38] flooding is limited to nodes within $k$ hops from the requester, where $k$ is the number of hops from the requester to the data center, but the overhead is still high. In a wireless network where nodes are uniformly distributed, on average there are $\pi k^2$ nodes within $k$-hops range of a mobile node. Therefore, $\pi k^2$ messages are needed to find a data item using this method. Moreover, when a message is broadcast in the
network, many neighbors will receive it. Even if the mobile node is able to identify and drop duplicated messages, each node still needs to broadcast the messages at least once to ensure full coverage. If a node has $c$ neighbors on average, the total number of messages needs to be processed is $c\pi k^2$. Although the message complexity is still $O(k^2)$, the constant factor may be very high, especially when the network density is high.

3.5.3 Cache Replacement Policies

Aggarwal et al. [4] classifies the existing cache replacement policies into three categories: direct-extension, key-based, and function-based. In the direct-extension category [49], traditional policies such as LRU or FIFO are extended to handle data items of non-homogeneous size. The difficulty with such policies in general is that they fail to pay sufficient attention to the data size. In the key-based policies [62], keys are used to prioritize some replacement factors over others; however, such prioritization may not always be ideal. Recently, function-based replacement policy has received considerable attention [4, 56, 70]. The idea in function-based replacement policies is to employ a function of different factors such as time since last access, entry time of the data item in the cache, transfer time cost, data item expiration time and so on. For example, the LNC-R-W3-U algorithm, proposed by Shim et al. [56], aims to minimize the response time in Web caching. Their cost function incorporates many system parameters such as the transfer time, the document size, and the invalidation rate. In [70], we proposed a generalized cost function for wireless environments. However, the solution was proposed for cellular networks and we did not consider mobility, which is different from this work.
3.6 Summary of Cooperative Cache Schemes

In this chapter, we designed and evaluated cooperative caching techniques to efficiently support data access in multi-hop based mobile environments. Specifically, we proposed three schemes: CachePath, CacheData, and HybridCache. In CacheData, intermediate nodes cache the data to serve future requests instead of fetching data from the data center. In CachePath, mobile nodes cache the data path and use it to redirect future requests to the nearby node which has the data instead of the faraway data center. HybridCache takes advantage of CacheData and CachePath while avoiding their weaknesses. Cache Replacement policies are also studied to further improve the cache performance. Simulation results showed that the proposed schemes can significantly reduce the query delay when compared to SimpleCache and significantly reduce the message complexity when compared to FloodCache.
Chapter 4

Data Access in Multi-hop based Mobile Environments: Data Replication

4.1 Introduction

In multi-hop based mobile environments, direct communication between any two mobile nodes is possible when they are within the radio range of each other, in which case we say that these two mobile nodes are neighbors. Otherwise, the mobile nodes communicate through multi-hop routing [46]. However, since mobile nodes move freely and mobile nodes and the wireless links are not stable, disconnections may occur frequently. If a network is divided into two partitions due to these reasons, mobile nodes in one of the partitions cannot access the data held by the mobile nodes in the other partition. Thus, data accessibility in multi-hop based mobile environments is lower than that in the conventional fixed networks.

Data replication has been widely used to improve data accessibility in distributed systems, and we want to apply this technique to ad hoc networks. By replicating data at mobile nodes which are not the owners of the original data, data accessibility can be improved because there are multiple replicas in the network and the probability of finding one copy of the data is high. Further, data replication can also reduce the query delay, since mobile nodes can get the data from some nearby replicas.
Generally speaking, data replication can increase the data accessibility and reduce the query delay if there are plenty of storage space in the mobile nodes. However, mobile nodes only have limited storage space, bandwidth and power, and hence it is impossible for one node to hold all the data considering these limitations. Therefore, it is important for mobile nodes to cooperate with each other; i.e., contribute part of their storage space to hold data of others. When a node only replicates part of the data, there will be a tradeoff between query delay and data accessibility. For example, replicating most data locally can reduce the query delay, but it reduces the data accessibility since many nodes may end up replicating the same data locally, while some data are not replicated by anyone. To increase the data accessibility, nodes should not replicate the same data that neighboring nodes already replicate. However, this solution may increase the query delay since some nodes may not be able to replicate the most frequently accessed data, and have to access it from neighbors. Although the delay of accessing the data from neighbors is shorter than that from the data owner, it is longer than accessing the data locally.

In this chapter, we propose data replication schemes that address both the query delay and the data accessibility. As both metrics are important for mobile nodes, our schemes need to balance the tradeoffs between data accessibility and query delay under different system settings and requirements. Simulation results show that the proposed schemes can achieve a balance between these two metrics and provide satisfying system performance.

The rest of this chapter is organized as follows. In the Section 4.2, a brief review of the related work is presented. Section 4.3 gives some preliminaries of the research.
Section 4.4 describes the proposed schemes in detail. Section 4.5 evaluates the proposed schemes through extensive simulations. Section 4.6 summarizes our research on data replication schemes in multi-hop based mobile environments.

4.2 Related Work

Data replication has been extensively studied in the Web environment [21, 39, 51, 71]. The goal is to place some replicas of web servers among a number of possible locations so that the query delay is minimized. In the Web environment, links and nodes are stable. Therefore, the performance is measured by the query delay, and data accessibility is not a big issue. These replication schemes work at the whole database level. That is, the whole database is replicated as a unit to one or more locations. It is more complex when replication is studied at data item level, i.e., how to replicate data items to various nodes with limited memory spaces.

Data replication has been studied in distributed database systems [21, 63, 64]. In such systems, nodes that host the database are more reliable and less likely to fail/disconnect than that in ad hoc networks. Therefore, a small number of replicas can be used to provide high accessibility. However, in ad hoc networks, node/link failure occurs frequently, and data accessibility becomes an important issue.

Several data replication schemes have been proposed in wireless networks [29, 50]. These schemes assume an environment where mobile nodes access database at sites in a fixed network, and create replicas of data on the mobile nodes because wireless communication is more expensive than wired communication. Their major concern is to keep the consistency between the original data and its replicas. The difference between
these schemes and ours is that our schemes are proposed in multi-hop ad hoc networks. There is no central server and data are fully distributed at mobile nodes.

Hara [26] proposed data replication schemes in ad hoc networks. These schemes are based on the intuition that to improve data accessibility, replicating the same data near neighboring nodes should be avoided. However, this intuition may not be valid when the link failure probability is taken into consideration. More detailed discussion will be presented in Section 4.4.1. Another drawback is that it only considers the accessibility, without considering the query delay. Both drawbacks will be addresses in this thesis to provide better data replication.

Some other researchers also addressed data access issues in ad hoc networks where network partition may occur. Karumanchi et. al. [36] and Luo et. al. [40] adopted the quorum based system to improve the data availability in ad hoc networks. Nuggehalli et. al. [43] proposed a POACH algorithm to deal with the cache placement problem so that the query delay and energy consumption can be reduced, but they did not consider link failure and the data are always available. Wang and Li [60] proposed schemes to deal with network partitions due to node movement by duplicating services in the network. Their schemes can provide guaranteed service with minimal number of duplicated services. Different from these previous works, we study the tradeoffs between the query delay and data accessibility.

Besides data replication, caching can also be used to improve data accessibility and reduce the query delay [13]. In [69], we proposed a cooperative cache based data access framework, which allows the sharing and coordination of cached data among multiple mobile nodes. In this solution, after a node sends a data request to the data
owner, the data owner sends the data back. Since the data may go through multi-hops before reaching the requester. Intermediate nodes may cache the data or the path to the data. Later, if some other nodes request for similar data, and the request goes through any of these intermediate nodes, they can return the data or the path to the data. As the number of hops reduces, the query delay also reduces. Although the proposed solution can reduce the query delay, there is a limitation on how much they can achieve. Generally speaking, these schemes are passive approaches, since the data are only cached after some nodes start to use it. To further increase the data accessibility and reduce the query delay, we study proactive data replication techniques in this thesis.

4.3 Preliminaries

4.3.1 System Model

The following notations are used in our discussion:

- $m$: the total number of mobile nodes.
- $N_i$: mobile node $i$.
- $n$: the total number of data items in the database.
- $d_i$: data item $i$.
- $s_i$: the size of $d_i$.
- $C$: the memory size of each mobile node for hosting data replicas.
- $f_{ij}$: the link failure probability between node $N_i$ and $N_j$. 
• $a_{ij}$: the access frequency of node $N_i$ to $d_j$.

The wireless network studied in this chapter has a total of $m$ nodes, $N_1, N_2, ..., N_m$. A database of $n$ items $d_1, d_2, ..., d_n$ is distributed in the network. At any given time, the link between $N_i$ and $N_j$ has a probability of $f_{ij}$ to fail. $f_{ij}$ is equal to $f_{ji}$ as we assume symmetric link conditions. The failed links may cause network partitions. Queries generated during the network partition time may fail because the requested data item is not available in the partition the requester belongs to. The access frequency of node $N_i$ to $d_j$ is $a_{ij}$. Each node maintains some amount of data locally and is called the original owner of these data. Each data item has one and only one original owner. For simplicity, we assume that data are not updated, and similar techniques used in [27] can be used to extend the proposed scheme to handle data update. To improve data accessibility, these data may be replicated to other nodes. Because of limited memory size, each node can only host $C, C < n$, replicas beside its original data. When a node $N_i$ needs to access a data item $d_j$, $N_i$ first searches its local memory. If $N_i$ cannot find a copy of $d_j$ in the local memory, $N_i$ communicates with its reachable nodes (through one-hop or multi-hop links) to get $d_j$. If the requesting node cannot communicate with any of the nodes that have $d_j$, $d_j$ is considered to be not accessible to $N_i$. Data Accessibility is defined as the number of successful data accesses over the total number of data accesses.

4.3.2 Problem Analysis

To simplify the problem, we first consider the optimization problem with only one performance metric. In this case, the goal of data replication is to allocate $n$ data items
among $m$ mobile nodes so that a certain performance metric (data accessibility, query delay, etc.) is optimized.

The optimal solution for this problem is not very practical due to the computational complexity. Let us use the optimization of the data accessibility metric as an example. It is obvious that replicating data improve the data accessibility. The performance improvement can be viewed as the profit of data replication. However, the profit of replicating $d_i$ at a certain node $N_j$ is affected by nodes’ access frequency to $d_i$, the network topology, and link failure probability. Therefore, the profit of $d_i$ is different at different nodes. Furthermore, the profit is also affected by previous replication of $d_i$ in the network. That is, the profit of allocating the first copy of $d_i$ in the network at $N_j$ is different from the case when allocating $d_i$ at $N_j$ but $d_i$ has already been replicated once, twice, or more at some other node(s). We can see that at one given node, the profit of a data item $d_i$ can take $(m - 1)!$ different values, where $m$ is the number of nodes, depending on the replication of $d_i$ at other nodes.

To further simplify this problem, let us assume that the profit of replicating $d_i$ at node $N_j$, denoted as $p(i, j)$, is not affected by the replication of $d_i$ at other nodes. This means that the profit of replicating $d_i$ can take $m$ different values. This is a special case of the Generalized Assignment Problem (GAP) [14, 41], which can be defined as:

**INSTANCE**: A pair $(B, S)$, where $B$ is a set of $m$ bins, and $S$ is a set of $n$ items. Each bin $j \in B$ has a capacity of $c(j)$, and for each item $i$ and bin $j$, we are given a profit $p(i, j)$ and a size $s(i, j)$.

**OBJECTIVE**: Find a subset $U \subseteq S$ of maximum profit such that $U$ has a feasible packing in $B$. 
In our case, the size of $d_i$ is fixed and the bin size is identical ($C$). However, Chekuri and Khanna [14] proved that even for the following special case, where

- each data item takes only two distinct profit values,

- each data item has an identical size across all bins and there are only two distinct item sizes, and

- all bin capacities are identical,

the problem is still APX-hard, which means that there exists some constant $\epsilon > 0$ such that it is NP-hard to approximate the problem within a factor of $(1 + \epsilon)$.

The analysis above shows that the data replication problem we studied is extremely hard in terms of the computational complexity. Even for a simplified version of the problem, it is still NP-hard to approximate the problem. Therefore, instead of trying to find a complex algorithm that is totally not practical to solve or approximate the problem, we present heuristics that can provide satisfying performance with very small computation overhead.

4.4 The Proposed Data Replication Schemes

4.4.1 An Example

Here we use an example to illustrate our ideas. Suppose we are studying a network with only two nodes $N_1$ and $N_2$. $N_1$ and $N_2$ may access four same-size data items $d_1, ..., d_4$ but each node only has enough space to host two data items. Similar to [26], we assume that the access probability of nodes to data items are available. These probabilities are listed in Table 4.1.
According to the DAFN (Dynamic Access Frequency and Neighborhood) scheme proposed by Hara [26], neighboring nodes should try to remove duplicated data items. In the first replication step, nodes replicate data that they are interested in. Therefore, both nodes replicate $d_1$ and $d_2$ in their memory. In the second step, when two neighboring nodes have the same data item $d_i$, the node that has a lower access probability to $d_i$ should replace $d_i$ with the next most interested data. Therefore, $N_1$ replaces $d_2$ with $d_3$ and $N_2$ replaces $d_1$ with $d_4$. The final replication result is: $N_1$ host $d_1$ and $d_3$ while $N_2$ should host $d_2$ and $d_4$.

Intuitively, DAFN is good because duplicated data are removed from neighboring nodes and the memory size is used more effectively. However, its data accessibility may be affected when the link failure probability is high. The average data accessibility for $N_1$ and $N_2$ is shown by line “DAFN” in Figure 4.1. If instead of replicating data according to DAFN, both $N_1$ and $N_2$ host $d_1$ and $d_2$, as shown by line “Our” in Figure 4.1, the data accessibility can be improved when the link failure probability is higher than 0.25. If the query delay is considered, the DAFN scheme obviously incurs higher query delay because many queries have to be satisfied by neighboring node.
Fig. 4.1. Data accessibility under different link failure probability between $N_1$ and $N_2$.

In this example, the simple solution outperforms the DAFN scheme proposed in [26] because DAFN does not consider two important factors: the link stability between mobile nodes and the query delay. In our schemes, we consider both factors when making replication choices. Due to the complexity of the problem, next, we present the heuristics used in our solution.

4.4.2 Heuristics

Because mobile nodes have limited memory space, it is impossible for them to hold all their interested data. They have to rely on other nodes to get some data. If mobile nodes only host their interested data, it is possible that some data are in every node while some other data are not replicated. Therefore, it is important for mobile nodes to contribute part of their memory to hold data for other nodes. This is some kind of cooperation between mobile nodes. The problem is to decide the amount of space
that a mobile node should contribute because bad cooperation may actually reduce the performance, as shown in the example above.

We have the following heuristics: For a mobile node, if its communication links to other nodes are stable, more cooperation with these nodes can improve the data accessibility; if the links to other nodes are not very stable, it is better for the node to host most of the interested data locally. The above heuristic mainly addresses the issue of data accessibility. For query delay, it is better to allocate data near the interested nodes. The degree of cooperation affects both the data accessibility and the query delay. Various schemes are proposed so that different performance targets can be achieved.

4.4.3 Data Replication Schemes

4.4.3.1 Greedy Schemes

One naive greedy data replication scheme is to allocate the most frequently accessed data items until memory is full. However, this naive scheme, referred to as Greedy, does not consider the size difference between different data items. The data size should be considered because smaller data require less memory size, thus replicating them can save the memory size for other data items. Therefore, a better greedy scheme is to calculate the access frequency value of $d_k$ by the following function:

$$AF_i(k) = \frac{a_{ik}}{s_k}$$

(4.1)
This greedy scheme, referred to as **Greedy-S**, let node $N_i$ repeatedly picks the data item with the largest $AF_i$ value from the data set that are not yet replicated at $N_i$ until no more data can be replicated to the memory.

**Performance Analysis:** For simplicity, the data size is assume to be the same in the analysis, i.e., $s_i = 1, i = 1, 2, ..., n$. Because the computational complexity of the optimal scheme, we give an upper bound of the data accessibility by using a super-optimal algorithm, similar to the approach used in [51]. The solution given by the super-optimal algorithm is not a tight upper bound. It may be better than optimal and it may not be feasible. However, it is too difficult to find the tight upper bound and this super-optimal algorithm can be used for performance comparison.

A node $N_i$ may have multiple one-hop neighbors. Assume that the probability of all links between $N_i$ and its neighbors fail is $f_{N_i}$. For the greedy scheme, $N_i$ hosts $C$ most frequently accessed data, suppose this set of data is $S_C$. Then the data accessibility for the greedy scheme, $A_{\text{greedy}}$, follows:

$$A_{\text{greedy}} \geq \sum_{d_k \in S_C} a_{ik} \quad (4.2)$$

A super optimal solution for $N_i$ would be allocating $C$ most frequently access data in $N_i$, but allocating the other data in a way that they are all accessible from $N_i$’s neighbors (this may not be possible in practice). It’s data accessibility, $A_{\text{super}}$, is

$$A_{\text{super}} = \sum_{d_k \in S_C} a_{ik} + (1 - f_{N_i}) \times \sum_{d_k \notin S_C} a_{ik} \quad (4.3)$$
Therefore,

\[
\frac{A_{\text{greedy}}}{A_{\text{super}}} \geq \frac{\sum_{d_k \in S_C} a_{ik}}{\sum_{d_k \in S_C} a_{ik} + (1 - f_{N_i}^*) \sum_{d_k \notin S_C} a_{ik}} \geq \sum_{k=1}^n \frac{a_{ik}}{a_{ik}}
\]

(4.4)

Fig. 4.2. The worst case data accessibility of greedy scheme compared to the super-optimal scheme under different memory size $C$ (the total number of data items $n = 100$)

**Numeric Result:** Figure 4.2 gives the numeric result that compares the worst case data accessibility of the greedy scheme with the data accessibility of the super-optimal scheme. The node access pattern is based on Zipf-like distribution [73], which has been frequently used [6] to model non-uniform distribution. In the Zipf-like distribution,
the access probability of the $k^{th}$ ($1 \leq k \leq n$) data item is represented as follows.

$$P_{a_k} = \frac{1}{k^\theta \sum_{i=1}^{n} \frac{1}{i^\theta}}$$

(4.5)

where $0 \leq \theta \leq 1$. When $\theta = 1$, it follows the strict Zipf distribution. When $\theta = 0$, it follows the uniform distribution. Larger $\theta$ results in more “skewed” access distribution. That is, more data accesses focus on the data items with small data id, which are called “hot” data.

We can see that the greedy scheme performs relatively well even when compared to a super-optimal scheme that may not be feasible at all. When the access pattern is more skewed, the greedy scheme performs better as more hot data access can be served by the replicated local copies.

One drawback of the greedy scheme is that it does not consider the cooperation between neighboring nodes and its performance may be limited. The following sections present schemes that include different degree of cooperation between neighboring nodes following our heuristics.

### 4.4.3.2 The One-To-One Optimization (OTOO) Scheme

In this scheme, each mobile node only cooperates with at most one neighbor to decide which data to host. Suppose node $N_i$ and $N_j$ are neighboring nodes. $N_i$ calculates the Combined Access Frequency (CAF) value of $N_i$ and $N_j$ to data item $d_k$
at $N_i$, denoted as $CAF_{1ij}$, by using the following function:

$$CAF_{1ij}(k) = (a_{ik} + a_{j} * (1 - f_{ij}))/s_i$$  \hspace{1cm} (4.6)$$

Similarly $N_j$ calculates its combined access frequency to $d_k$ with the following function:

$$CAF_{1ji}(k) = (a_{jk} + a_{i} * (1 - f_{ij}))/s_i$$  \hspace{1cm} (4.7)$$

Each node sorts the data according the CAF1 value and picks data items with the highest CAF1 values to replicate in its memory until no more data items can be replicated. The CAF1 value function is designed so that 1) it considers the access frequency from a neighboring node to improve the data accessibility. 2) it considers the data size. If other criteria are the same, data with smaller size is given a higher priority for replicating because they can improve the performance while reducing the memory size requirement. 3) it gives the accessibility from the node itself a high priority so that its interested data can be replicated locally to improve the data accessibility and the query delay. The OTOO scheme works as follows:

1. All nodes are marked as "white" initially, which means that no one has executed the allocation process yet. These nodes broadcast their ids and their access frequency for each data item.

2. Among the white nodes, the node which has the smallest id among its neighboring white nodes starts the following process. It sends an invitation to the neighboring white node with which it has the lowest link failure probability. If the neighbor
only receives one such invitation, these two neighboring nodes calculate the CAF1 values and each node allocates data items with the highest CAF1 values until it cannot accommodate more data. Then both nodes are marked as “black” and no longer participate the replication process until the next allocation period.

3. In case that two or more nodes start the process at the same time, as long as they do not pick the same node as the most reliable neighbor, they can allocate their replicas at the same time. Otherwise, the node picked by more than one neighbor only accepts the invitation from the node with the lowest id. All other inviting nodes have to select another neighbor again.

4. If all neighbors of a white node are black nodes, which means that this white node cannot find any neighbor to cooperate in the allocation process, it only allocates its own most interested data items to its memory.

It is possible that according to OTOO, node $N_i$ should host $d_j$ but $N_i$ is separated from nodes that have $d_j$ because of network partitions. In this situation, $N_i$ selects the next best candidate (data item) according the replication scheme. This rule is also applied to other replication schemes proposed in the following.

4.4.3.3 The Reliable Neighbor (RN) Scheme

OTOO considers neighboring nodes when making data replication choices. However, it still considers its own access frequency as the most important factor because the access frequency from a neighboring node is reduced by the link failure probability factor. To further increase the degree of cooperation, we propose the Reliable Neighbor
(RN) scheme that contributes more memory to replicate data for neighboring nodes. In this scheme, part of a node’s memory is used to hold data for its Reliable Neighbors. For node $N_i$, a neighboring node $N_j$ is considered to be $N_i$’s reliable neighbor if

$$1 - f_{ij} > T_r,$$

where $T_r$ is a threshold value. Let $nb(i)$ be the set of the $N_i$’s reliable neighbors. The total contributed memory size $N_i$, denoted as $Cc(i)$, is set to be

$$Cc(i) = C \times \min(1, \sum_{N_j \in nb(i)} (1 - f_{ij})/\alpha)$$

(4.8)

where $\alpha$ is a system tuning factor.

Intuitively, $N_i$ contributes more memory if its links with neighboring nodes are more stable. The two extreme cases are: 1) when $Cc(i) = C$, $N_i$ contributes all of its memory to hold data for neighboring nodes; 2) when $f_{ij} = 1, \forall N_j \in nb(i)$, $N_i$ does not contribute any memory. The reason behind the RN scheme is that when links to neighboring nodes of $N_i$ are stable, it is OK for $N_i$ to hold more data for neighboring nodes as they also hold data for $N_i$. Because links are stable, such cooperation can improve the data accessibility. If links are not stable, data on neighboring nodes have low accessibility and may incur high query delay. Thus, cooperation in this situation cannot improve data accessibility and nodes should be more “selfish” in order to get better performance.
The data replication process works as follows. Node $N_i$ first allocates its most interested data to its memory, up to $C - Cc(i)$ memory space. Then all the rest of the data are sorted according to $CAF2$ to a list called the neighbor’s interest list. The $CAF2$ value of $N_i$ to $d_k$ is defined as:

$$CAF2_i(k) = \left( \sum_{N_j \in nb(i)} a_{jk} \times (1 - f_{ij}) \right) / s_k$$

(4.9)

$Cc(i)$ memory space are used to allocate data with highest $CAF2_i$ values. There may be some overlap between $N_i$’s interested data and the allocated data interested by $N_i$’s neighbors. If during the allocation, a data item is already in the memory, this data item will not be allocated again and the next data item on the neighbor’s interest list is chosen instead.

### 4.4.3.4 Computation Overhead of the Proposed Schemes

The proposed schemes need to sort all the data items according to the CAF value. The computational complexity of sorting them is $O(n \log n)$. This is the same as that of the DAFN scheme, although the constant factor may be higher because the calculation of the CAF value. However, as shown in the following section, our schemes are able to provide much better performance than the DAFN scheme.
4.5 Performance Evaluation

In this section, we evaluate the performance of the proposed schemes: OTOO, RN2 (RN with $\alpha = 2$), RN8 (RN with $\alpha = 8$), RN16 (RN with $\alpha = 16$), Greedy-S, by comparing to the DAFN scheme proposed by Hara [26] and the Greedy scheme.

4.5.1 The Simulation Model and System Parameters

In the simulation, $m$ nodes are placed randomly in a $1500m \times 1500m$ area. The radio range is set to be $D$. If two nodes $N_i$ and $N_j$ are within the radio range, i.e., the distance between them $D(i, j) < D$, they can communicate with each other. However, the link between them may fail. The link failure probability $f_{ij}$ is defined as

$$f_{ij} = \left(\frac{D(i, j)}{D}\right)^2$$ (4.10)

Equation (4.10) is adopted according to the fact that the wireless signal strength decreases with a rate between the order of $r^2$ and $r^4$, where $r$ is the distance to the signal source. Note that proposed schemes do not depend on the failure model in Equation (4.10). They are able to work as long as the failure probability between neighboring nodes can be estimated. Because the link failure may be caused by many factors such as channel condition, node movement and node failure, the actual link failure probability may not be estimated accurately. In the simulation, we also study the effect of inaccurate link failure probability on the proposed schemes. This is done by introducing an estimation error factor $\delta$. The estimated link failure probability is the actual link failure probability multiplied by a factor, which is exponentially distributed with a mean value.
of $\delta$. Different $\delta$ values, ranging from 0.6 to 1.4 are used to study our schemes’ sensitivity to inaccurate estimations.

Similar to [26], the number of data items $n$ is set to be the same as the number of nodes $m$. Data item $d_i$’s original host is $N_i$, for all $i \in [1, m]$. The data item size is uniformly distributed between $s_{\text{min}}$ and $s_{\text{max}}$ memory units. Each node has a memory size of $C$. Different $C$ values are studied in the simulation to show their effects on the system performance.

Two access patterns are used in the simulation.

1. All nodes follow the Zipf-like access pattern, but different nodes have different hot data. This is done by randomly selecting an offset value for each node $N_i$: $\text{offset}_i$, which is between 1 and $n - 1$. The actual access probability of $N_i$ to data item $d_k$ is given by:

$$P_{a_k} = \frac{1}{(((k + n - \text{offset}_i)\%n + 1)^{\theta} \sum_{j=1}^{n} \frac{1}{j^{\theta}})}$$

This means that the most frequently accessed data item $id$ is moved to be $\text{offset}_i$ instead of 1 as given in Equation (4.5); the second frequently accessed data item $id$ is $\text{offset}_i + 1$ instead of 2, and so on.

2. All nodes have the same access pattern and they have the same access probability to the same data item.

The performance metrics used in the simulation are data accessibility and query delay. When a query for data $d_k$ is generated by node $N_i$, if $d_k$ can be found at a node that is reachable through single or multi-hop links, this access is considered successful
and the query delay is the number of hops from $N_i$ to the nearest node that has $d_k$. If $d_k$ is in the local memory of $N_i$, the query delay is 0. The average query delay only considers successful accesses, i.e., it is the total delay of successful accesses divided by the total number of successful accesses.

Most system parameters are listed in Table 4.2. The second column lists the default values of these parameters. In the simulation, we may change the parameters to study the impacts of these parameters. The ranges of the parameters are listed in the third column.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default value</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of nodes $m$</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Number of data items $n$</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>$s_{\text{min}}$</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>$s_{\text{max}}$</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Memory size $C$</td>
<td>20</td>
<td>10 - 40</td>
</tr>
<tr>
<td>Radio range $D$</td>
<td>250m</td>
<td>50m - 300m</td>
</tr>
<tr>
<td>Zipf parameter $\theta$</td>
<td>0.6</td>
<td>0.2 - 1.0</td>
</tr>
<tr>
<td>$T_F$</td>
<td>0.6</td>
<td>0.2 - 0.8</td>
</tr>
<tr>
<td>Error factor $\delta$</td>
<td></td>
<td>0.6 - 1.4</td>
</tr>
</tbody>
</table>
4.5.2 Simulation Results

4.5.2.1 Fine-tuning the RN scheme

In Figure 4.3, we evaluate the effects of $T_r$, which affects the number of cooperative neighbors in the RN scheme. Larger threshold value $T_r$ results in smaller number of cooperative neighbors, and vice versa. We can see that $T_r$ has the largest effect on the performance of RN2, which is because RN2 contributes the largest portion of the memory size to neighbors. We found that $T_r = 0.6$ achieves a balance between the data accessibility and query delay, and similar results were found when nodes have different access pattern. Therefore, $T_r = 0.6$ is adopted in the following.

Fig. 4.3. Performance of RN as a function of $T_r$ when nodes have the same access pattern
4.5.2.2 Effects of Zipf Parameter ($\theta$)

In this section, we evaluate the effects of the Zipf parameter $\theta$ on the system performance. As $\theta$ increases, more accesses focus on hot data items and the data accessibility is expected to increase.

![Graph showing data accessibility and query delay as functions of Zipf parameter $\theta$.]

Fig. 4.4. Performance as a function of $\theta$ when nodes have different access pattern

Figure 4.4 demonstrates the effects of the Zipf parameter $\theta$ on the system performance when nodes have different access pattern. Figure 4.4 (a) shows that the proposed schemes outperform the DAFN scheme in terms of data accessibility in almost all cases. This is because: first, our schemes consider the link failure probability when replicating data; second, our schemes avoid replicating data items that are not frequently accessed by using the CAF value. On the other hand, the DAFN scheme does not consider the
link failure probability and it sometimes replicates data items with low access frequency instead of frequently accessed data items, as shown in the example in Section 4.4.1.

Figure 4.4 (b) shows the query delay of different schemes. The DAFN scheme is outperformed by the proposed schemes in all situations. This shows that our schemes can achieve better performance in terms of both data accessibility and query delay. The DAFN scheme tries to avoid duplicated data items among neighboring nodes, which means that even if a data item is popular among two neighboring nodes, it is still allocated at only one of the neighboring nodes. Therefore, many accesses have to be satisfied by querying neighboring nodes, which increases the query delay.

From Figure 4.4 (b), we can also find that the relation of query delay is $R_{N2} > R_{N8} \approx R_{N16} > OTOO$. This shows that when nodes have different interest, it is better for them to host data they are interested in, and cooperation among them does not show significant advantages.

Fig. 4.5. Performance as a function of $\theta$ when nodes have the same access pattern
Figure 4.5 shows the effects of the Zipf parameter $\theta$ on the system performance when nodes have the same access pattern. Note that in this situation, Greedy-S and OTOO are the same because Equation (4.2) only differs from Equation (4.6) at a constant factor. Greedy-S performs better than Greedy because it gives higher priority to data items with smaller size, and thus more important data can be replicated and the performance is improved.

We can see from Figure 4.5 that all the proposed schemes perform much better than the DAFN scheme in terms of data accessibility and all the proposed schemes except RN2 perform better than DAFN in terms of query delay. Comparing RN2, RN8, RN16, and OTOO, we find that the relation of their data accessibility is $RN2 > RN8 > RN16 > OTOO$ (RN2 performs the best) while the relation of their query delay is $RN2 > RN8 > RN16 > OTOO$ (OTOO performs the best). This clearly shows the tradeoffs between these two performance metrics. Higher degree of cooperation improves the data accessibility, but it also increases the query delay because more data need to be retrieved from neighboring nodes. This figure also gives us directions about how to achieve certain performance goals. If a high data accessibility is required, nodes should be more cooperative with neighboring nodes so that more data can be replicated in the network. If a low query delay is required, nodes should be more “selfish” so that requests can be served locally instead of by neighboring nodes.

4.5.2.3 Effects of Radio Range $D$

Figure 4.6 and 4.7 show the effects of the radio range on the system performance. When the radio range increases, the network is better connected and the data accessibility
Fig. 4.6. Performance as a function of the radio range when nodes have different access pattern
is expected to increase. Figure 4.6 (a) shows that all schemes perform as expected. The proposed schemes perform much better than DAFN when the radio range is small. When the radio range is very large, different schemes have similar data accessibility. This is because the network partition is very rare in this situation and most data can be found in a reachable node.

Figure 4.6 (b) shows that the query delay increases as the radio range increases. This is because when the network is better connected, some data that are previously not available can now be found at faraway nodes, which increases the average query delay. The proposed schemes always result in lower query delay than the DAFN scheme. When the radio range is extremely small, the query delay of all scheme reduces to near zero, since it is hard to find a neighbor with such small radio range and almost all requests are served locally.

Figure 4.6 (c) evaluates the replication overhead in terms of the amount of data traffic incurred by the data replication schemes. The Greedy scheme and the Greedy-S scheme generate the lowest replication traffic because in their schemes, nodes do not cooperate with their neighbors. Because DAFN tries to remove duplicated data items in neighboring nodes, it always incurs the highest traffic. Among the RN schemes, RN2 generates the highest traffic and RN16 generates the lowest. This is because RN2 contributes a large amount of memory space to neighboring nodes but RN16 contributes the smallest. Overall, all of the proposed schemes perform better than DAFN in terms of the replication traffic. From Figure 4.6 (a), (b), and (c), we can conclude that our schemes can provide better performance with low traffic overhead.
Figure 4.7 shows the effects of radio range when nodes have the same access pattern. RN2 has the highest data accessibility, and it also has the highest query delay in most cases. The comparison result is similar to that in Figure 4.5 when the Zipf parameter $\theta$ changes.

4.5.2.4 Effects of Memory Size $C$

In this section, we evaluate the system performance when the memory size ($C$) changes. As $C$ increases, more data can be hosted by a node and the data accessibility increases. Similarly, more data can be found locally as $C$ increases and the query delay decreases.

Figure 4.8 shows that when nodes have different access patterns, the proposed schemes increase the data accessibility while providing lower query delay compared to the DAFN scheme. The difference of data accessibility is not very high because when
Fig. 4.8. Performance as a function of memory size when nodes have different access pattern

Fig. 4.9. Performance as a function of memory size when nodes have the same access pattern
nodes have different access pattern, they can simply replicate their interested data locally to achieve a high data accessibility. Thus the room for improvement is small. When nodes have the same access pattern, Figure 4.9 shows that our schemes perform much better than DAFN in terms of data accessibility while still results in low query delay.

4.5.2.5 Effects of the Error Factor of Link Failure Estimation ($\delta$)

![Data Accessibility](image1)

![Query Delay](image2)

Fig. 4.10. Performance as a function of $\delta$ when nodes have different access pattern

This section evaluates the effects of error factor in link failure probability estimation, $\delta$. The performance of DAFN, Greedy, and Greedy-S is not affected by $\delta$ as they do not depend on the estimation of link failure probability. From Figure 4.10 and 4.11, we can see that although $\delta$ affects the performance of RN2, RN8, RN16, and OTOO,
Fig. 4.11. Performance as a function of $\delta$ when nodes have the same access pattern the effect is not very significant even when the error is very large. We can conclude that they are robust and not sensitive to estimation errors.

4.6 Summary of Data Replication Schemes

In multi-hop based mobile environments, network partitions are common and data accessibility is low. In this chapter, we proposed several data replication schemes to deal with this issue. In the OTOO scheme, nodes cooperate with only one neighboring node when making data replication decision. In RN2, RN8, and RN16, nodes cooperate with more neighboring nodes and contribute more memory space to hold data for neighboring nodes. The difference between our scheme and existing schemes such as DAFN is that our schemes take link failure into account during data replication and try to balance data accessibility and query delay. Extensive performance evaluations demonstrate that
the proposed schemes can provide high data accessibility. At the same time, our schemes achieve a balance between data accessibility and query delay.
Chapter 5

Conclusions and Future Work

This chapter concludes our thesis and discusses some future work.

5.1 Conclusions

In this thesis, power-aware data access schemes in different mobile environments have been studied. Prefetch schemes have been designed and evaluated for single-hop based mobile environments, cooperative caching and data replication schemes have been designed and evaluated for multi-hop based mobile environments. These research studies have the same goal: to provide efficient data access to mobile nodes.

In single-hop based mobile environments, we proposed to use prefetching to improve system performance. We have proposed a value-based prefetch scheme (VP) that considers various factors such as data size, data access rate, and data update rate. Through theoretical analysis, we have proved that data with a high value can benefit mobile nodes more in terms of access cost. Thus, mobile nodes should give them higher priority during prefetching. In real situations when system parameters such as data access rate are not directly available, we have proposed different practical methods to estimate them based on the access history and the characters of different parameters. Because prefetching consumes battery power, mobile nodes should not prefetch data too aggressively. To determine how many data to prefetch, adaptive value-based prefetch
schemes (AVP) have also been proposed. These AVP schemes can either adapt the prefetch to reach a target battery life, or adapt the prefetch based on the remaining power level to achieve better performance.

With no support from a base station, mobile nodes face more challenges when accessing data in multi-hop based mobile environments. Existing schemes such as cooperative caching and data replication schemes designed for a wired network may not be directly applied to this environment because wireless mobile networks are not stable and mobile nodes have limited resources.

To address these issues, we have designed and evaluated cooperative caching schemes for multi-hop based mobile environments. In CacheData, mobile nodes cache passing-by data to serve future requests. In CachePath, mobile nodes cache the data path so that future requests can be redirected to a nearby node. Through theoretical analysis and simulation, we showed that both schemes have their advantages and disadvantages. To utilize their advantages while avoiding their disadvantages, we have designed the HybridCache scheme that combines these two schemes. The cache invalidation scheme is very important when a cache is used. We have adopted the TTL approach for cache invalidation. This scheme incurs very low overhead in our data access model. Overall, the proposed schemes can significantly improve the performance compared to traditional cache schemes and incur much lower overhead compared to message flooding based schemes. Therefore, they are suitable for resource-scarce mobile environments.

Data replication schemes have been studied to address the network partition issue in multi-hop based mobile environments. We investigated the data replication problem
through theoretical analysis and showed that this problem is a special case of the Generalized Assignment Problem, which is NP-hard to even approximate the problem. Then we designed several data replication schemes based on the heuristic that more cooperation should be employed when the neighbors are stable, and vice versa. The simulation results showed the tradeoffs between data accessibility and the query delay, and the proposed schemes perform much better than the existing schemes while achieving a balance of data accessibility and query delay.

5.2 Future Work

The thesis work can be extended in the following directions:

1. In single-hop based mobile environments, although we explored various adaptive approaches, many issues still need further investigation. For example, the channel state information [22] can be used when making prefetch decisions, i.e., $N_p$ can be dynamically adjusted based on the channel state. If the target battery life time is not known or only known with some probability, AVP_T needs to be extended to factor into these uncertainties. If the battery recharge cycle or the user profile is known or known with a high probability, how to enhance AVP_T and AVP_P needs further investigation.

2. In multi-hop based mobile environments, some issues related to data caching/replication may need to be addressed.

- In the cooperative caching schemes, we adopted the TTL approach to maintain a weak cache consistency. In some situations such as the battle field,
strong cache consistency is needed. Current schemes to maintain strong cache consistency, such as IR/UIR schemes, may incur high overhead in multi-hop based mobile environments. Thus, low-cost techniques to maintain cache consistency can greatly strengthen wireless data services and complement our cooperative caching schemes in multi-hop based mobile environments.

- We did not consider security issues in the proposed cooperative caching/data replication schemes. When data are shared in the mobile network, one of the major concerns is the duplication of sensitive data that the owner might want to restrict the access. To address this issue, we may need to define different levels of security for the data, with regard to duplicating and storing them in mobile nodes. The data server/owner can specify the level of security for each data item. Based on the security level, some data may not be cached/replicated, or only to a limited number of nodes. In the case of some most sensitive data, the data server only sends the encrypted version to a number of nodes. Those trusted nodes are able to get a shared key from the data server to decrypt the data. Further study is needed to design a mechanism that provides the owner a way to control the scope of caching, and yet does not undermine its flexibility.
References


Vita

Liangzhong Yin was born in Anhui, China. He received the B.E. degree and the M.E. degree in computer science and engineering from the Southeast University, Nanjing, China, in 1996 and 1999, respectively. From 1999 to 2000, he was an engineer in the Global Software Group of Motorola. He enrolled in the Ph.D. program in Computer Science & Engineering at The Pennsylvania State University in August, 2000. He is a student member of IEEE.

PUBLICATIONS


