Peer-to-Peer Cooperative Caching in Mobile Environments

Chow Chi-Yin

A thesis submitted in partial fulfillment of the requirements for the Degree of Master of Philosophy

Department of Computing
The Hong Kong Polytechnic University

December 2004

Certificate of Originality

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

Chow Chi-Yin (Name of student)

Abstract

Caching is a key technique for improving the data retrieval performance of mobile clients, who will store frequently needed data items in their local cache, often of a limited size. The emergence of the state-of-the-art peer-to-peer communication technologies now brings to reality what we call "cooperative caching" in which mobile clients can help one another in caching. They not only can retrieve data items from mobile support stations, but also from the cache in their peers, realizing a new dimension for mobile data caching. This thesis proposes a COoperative CAching scheme, called COCA, which can be tailored for pull-based, push-based and hybrid mobile environments. COCA was found to improve the access latency of client requests and the amount of relatively expensive requests forwarded to the server in the pull-based environment. It can also effectively reduce power consumption in the push-based and hybrid environments. We propose a cache signature scheme for the mobile clients to provide hints for them to determine whether to search the cache of their peers or directly enlist the server for help. We observe the need for cooperating peers to cache useful data items together, so as to further improve cache hit from peers. This could be realized by capturing the data requirement of individual peers in conjunction with their mobility patterns, for which we respond with two *group-based cooperative caching schemes* for mobile clients: *centralized* and *distributed* group-based COCA schemes or CGCoca and DGCoca respectively. We define a *tightly-coupled group* (TCG) as a collection of peers that possess similar mobility pattern and display similar data affinity. Built upon the COCA system, we propose a family of centralized and distributed algorithms to discover and maintain all TCGs dynamically in CGCoca and DGCoca respectively. Two cooperative cache management protocols: *cooperative*

cache admission control and *cooperative cache replacement*, are proposed to improve data accessibility in TCGs. We conduct performance studies of our COCA schemes based upon simulated experiments. The group-based COCA schemes are shown to outperform the conventional caching scheme, standard COCA and COCA with cache signature scheme. In the group-based COCA schemes, DGCoca is also found to perform better than CGCoca, as DGCoca is more effective in discovering and maintaining TCGs in mobile environments.

Acknowledgments

I would like express my deepest gratitude to my research supervisor Dr. Leong Hong Va for his invaluable support, advice, insight and guidance throughout this interesting and challenging research project. In addition, I would like to thank my great research group members, Dr. Alvin Chan, Mr. Ken Lee, Mr. Gary Lam and Miss Jing Zhou for their suggestions and sharing experiences and knowledge with me.

Last but not the least, I also would like to thank my family - my elder sister Aster Chow and my parents, and my best friend Miss Flora Lee for their endless love, support and encouragement. This thesis is dedicated to all of them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Caching is a key technique for improving the data retrieval performance of mobile clients. The recent widespread deployment of new peer-to-peer (known as P2P throughout the thesis) wireless communication technologies, such as IEEE 802.11 [49] and Bluetooth [16], coupled with the fact that the computation power and storage capacity of most mobile devices have been improving at a fast pace, a new information sharing paradigm, known as P2P information access has rapidly taken shape. The mobile clients can communicate among themselves to share information rather than having to rely solely on their connections to the server. This paradigm of sharing cached information among the mobile clients that assist one another is called *mobile cooperative caching*.

1.1 Background

In this section, we will give a brief description on the background of wireless information access, mobile systems and mobile cooperative caching.

1.1.1 Infrastructure- and Ad-hoc-based Wireless Information Access

Mobile computing environments can be classified into two different types of architectures, *infrastructure-* and *ad-hoc-based*. An infrastructure-based mobile system is formed with a wireless network connecting mobile hosts (MHs) and mobile support stations (MSSs). MHs are clients equipped with portable devices, such as laptops, PDAs, cellular phones and so on, while MSSs are stationary servers providing information access for the MHs residing in their predefined service areas. This kind of networking architecture, with the presence of MSSs, is known as an infrastructure-based network. It is the most commonly deployed one in real life. Furthermore, the emergence of the state-of-the-art P2P communication technologies leads to the development of an ad-hoc-based mobile computing architecture, called mobile ad-hoc network (MANET). In MANETs, the MHs can share information among themselves without any help of the MSSs. This kind of sharing paradigm is also referred to as a P2P data dissemination model.

1.1.2 Data Dissemination Models in Mobile Environments

In the infrastructure-based network, the MHs can retrieve the required data items from MSSs, either by requesting them over shared point-to-point channels, i.e., pull-based data dissemination model, or catching them from scalable broadcast channels, i.e., push-based data dissemination model, or through the utilization of both types of channels, i.e., hybrid data dissemination model. In MANETs, the MHs are able to perform P2P information access among themselves without the presence of MSSs. These four types of data dissemination models will be briefly described in the following sections.

1.1.2.1 Pull-based Data Dissemination Model

In a pull-based data dissemination model, when an MH cannot find the required data item in its cache, i.e., a cache miss, it sends a request to the MSS via an uplink channel. The MSS processes the request and then sends the required data item back to the MH via a downlink channel. The drawback of this model is that as mobile environments are characterized by asymmetric communication, in which the downlink channels are of much higher bandwidth than the uplink channels, the MSS would potentially become a scalability bottleneck in the system, as it serves an enormous number of MHs [3].

1.1.2.2 Push-based Data Dissemination Model

It is also known as a data broadcast model, in which the MSS periodically broadcasts the data items to the MHs residing in its service area via a broadcast channel. The MHs need not communicate with the MSS explicitly. When they encounter cache misses, they listen to the broadcast channel and catch the required data items when the items appear in the channel. By comparison with the pull-based model, data broadcast model is more scalable, as its performance does not depend on the number of MHs. However, since the data items are broadcast sequentially, the access latency gets longer with increasing number of data items being broadcast. The frequency of data items being broadcast in a broadcast cycle is determined by different scheduling schemes: flat broadcast, probabilistic-based broadcast, broadcast disks and on-demand data broadcast.

1. **Flat Broadcast.** It is the simplest broadcast scheduling scheme. The data items are broadcast with same frequency to the MHs. Thus, the expected access latency for every data item is the same, i.e., half of the broadcast cycle. However, this scheme does not consider the data access patterns, so that it gives

poor performance compared with other schemes when the data access patterns are skewed.

2. **Probabilistic-based Broadcast.** In this scheduling scheme, the frequency of the data items to be broadcast is based on their access probabilities [89]. For data item, i, the probability of that i is selected to be broadcast is $p_i = \frac{\sqrt{q_i}}{\sum_{j=1}^{N} \sqrt{q_j}}$, where q_j is the access probability for data item j and N is the total number of data items being broadcast. Although it improves the average performance when the data access patterns are skewed, the MHs suffer from longer access latency in obtaining data items with low access probabilities.

3. **Broadcast Disks.** The MSS maintains multiple broadcast disks with different disk sizes [2]. In broadcast disks, the smaller the disk size the higher the speed of the disk spins at. The data items allocated to the broadcast disks with smaller disk size enjoy a higher broadcast frequency. Every data item is assigned to one of the disks based on the data access probability. The data items with higher access probabilities are allocated to the disk with higher spinning speed, so that those data items can be broadcast more frequently. Broadcast disks scheduling scheme improves the system performance with skewed data access patterns. However, it suffers from drawbacks similar to the probabilistic-based broadcast scheduling. The data access latency increases, as the number of data items being broadcast gets larger, or when the MHs access the data items with low access probabilities.

4. **On-demand Data Broadcast.** The previous broadcast scheduling schemes exhibit poor performance, when the total number of data items becomes large. To alleviate this problem, on-demand data broadcast scheduling schemes are proposed [4, 6]. The MSS reserves some of channels as uplink channels for

the MHs to send their requests. The MSS stores the requests in a buffer and schedules the requests from it by different selection policies, e.g., FIFO (First-In, First-Out), MRF (Most Requests First), LWF (Longest Wait First) or semantic-based broadcast schemes [56, 57], etc. Then, the MSS broadcasts the required data items to the MHs via broadcast channels. The answered requests are then removed from the buffer.

1.1.2.3 Hybrid Data Dissemination Model

To overcome the downside of pull- and push-based data dissemination models, a hybrid data dissemination model, which encompasses both models, is proposed [3, 37, 47, 60, 78]. In the hybrid data dissemination model, part of the bandwidth is reserved for broadcasting data items to the MHs, and the rest is dedicated to pull-based communication. In the hybrid data dissemination environment, the system architects have to deal with two major issues: the bandwidth allocation between broadcast and uplink channels, and data allocation, i.e., which data items should be allocated to the broadcast channel. The details of the bandwidth and data allocation will be described in Chapter 3.

1.1.2.4 Peer-to-Peer Data Dissemination Model

This data dissemination model is also known as a mobile ad-hoc network (MANET), in which there is no dedicated infrastructure support, and an MH works as both a client, server and even as a router. In MANETs, an MH can directly communicate with other MHs residing in its transmission range; however, it has to communicate with other MHs not residing in the range through multi-hop routing [17]. MANET is practical with many environments with no stationary MSSs, such as battlefield, rescue operations [33], etc. In MANETs, the MHs can move freely, and disconnect

themselves from the network at any time. These two characteristics lead to dynamic network topology changes. As a result, the MHs may suffer from long access latency or access failure, when the peers holding the required data items are far way or unreachable. The latter situation could be caused by network partitioning [85].

1.1.3 Unique Properties of Mobile Systems

A mobile system possesses a lot of unique properties and limitations that make it more than simply a distributed system, but a fertile area of research [12, 33, 50, 90]. The essential properties of mobile systems are:

- **Asymmetric Communication.** Mobile environments are characterized by asymmetric communication, in which the downlink channels are of much higher bandwidth than the uplink channels. The limited bandwidth in uplink channel could become the system bottleneck, when the MSS serves an enormous number of MHs.

- **Client Mobility.** There is no physical linkage between MHs and MSSs; the MHs can freely roam around, so it is difficult to locate or predict their locations, and the unrestricted mobility also causes handover between MSSs or network partitioning in MANETs.

- **Frequent Disconnection.** The MHs frequently disconnect themselves from the network voluntarily (to save energy) or involuntarily (handover or network failures).

- **Limited Battery Power.** The battery power on mobile devices is limited. Thus, power consumption is a crucial factor on designing a data dissemination model and a set of communication protocols.

1.2 Mobile Cooperative Caching

In a mobile cooperative caching environment, the MHs can obtain their required data items not only from the MSSs, but also from the cache of their peers. COCA is appropriate for an environment in which a group of MHs possesses a common access interest. For example, in an exhibition, the MHs crowded near the same booth are probably interested in information related to the booth. Also, in a shopping mall, MHs residing in the same shop are likely to access information pertaining to the shop. Furthermore, at a conference, the participants staying in the same session room also likely possess common research interest. For instance, they may download similar documents of the proceedings or related works from the server. When the MHs share a common access interest, there is a higher probability for them to find the required data items from their peers.

Consider an MH, m, with a set of neighboring peers, $\mathcal{P}(m)$, where m can communicate with $\mathcal{P}(m)$ via P2P communication. Let A_m be the set of accessed data items by m, C_m be the set of data items cached and P_m be the set of data items cached by its peers $\mathcal{P}(m)$. Thus, $P_m = \cup_{i \in \mathcal{P}(m)} C_i$. The local cache hit ratio $Hit_L(m)$ is $\frac{|A_m \cap C_m|}{|A_m|}$ and global cache hit ratio $Hit_G(m)$ is $\frac{|(A_m \cap P_m) - (C_m \cap P_m)|}{|A_m|}$. The overall cache hit rate $Hit(m) = Hit_L(m) + Hit_G(M) = \frac{|A_m \cap (C_m \cup P_m)|}{|A_m|}$. COCA can achieve better system performance, as $Hit_G(m)$ is normally larger than zero, i.e., $(A_m \cap P_m) - (C_m \cap P_m) \neq \emptyset$. To obtain a higher Hit_G, m and its peers $\mathcal{P}(m)$ are expected to own similar access set $(A \cap P)$ and should try to cache more *distinct* data items in the set $(C \cup P)$. To control the data replicas in the system, two cooperative cache management protocols: cooperative cache admission control and cooperative cache replacement, are proposed in Chapter 5.

COCA brings several benefits to the MHs and the MSS. In some cases, the telecommunication service providers charge the users at a unit cost $Cost$ per access when they

access remote data items in the mobile system. The cost can be based on the number of sessions, connection time or transmission volume. The users have to pay for the cost of $|A| \times (1 - Hit_L) \times Cost$. With COCA, the users can save a cost of $|A| \times Hit_G \times Cost$. Likewise, the number of access misses as a result of an MH moving outside of the service area can be reduced by the amount of Hit_G, since it also has a chance to obtain the required data items from its peers. Furthermore, the server workload can be reduced from $NumClient \times |A| \times (1 - Hit_L)$ to $NumClient \times |A| \times (1 - Hit_L - Hit_G)$, where $NumClient$ is the number of MHs in the system. This desirable feature can be considered as a kind of indirect load sharing technique in mobile environments. Based on our simulated experiment results, COCA can reduce access latency and server workload in the pull-based environment, and improve access latency, server workload and even the power consumption in the push-based and hybrid environments.

1.3 Motivation

To support effective communication in mobile environments, a robust, scalable, adaptive communication platform is necessary. As mobile environments are characterized by asymmetric communication, in pure pull-based data dissemination model, the uplink channel would potentially be a scalability bottleneck in the system, as it serves an enormous number of MHs [3]. Although the push-based and hybrid data dissemination models are scalable, the MHs in these two models generally suffer from longer access latency and consume much more power than in the pull-based one, as they need to tune in to the broadcast channel, and wait for the broadcast channel index and the desired data items.

We now consider the other kind of communication architecture, MANET. MANET

is practical to a mobile system with no fixed infrastructure. However, it is not suitable for commercial applications alone. In MANETs, the MHs can rove freely, and disconnect themselves from the network at any instant. These two characteristics lead to dynamic network topology changes. As a result, the MHs may suffer from long access latency or access failure, when the peers holding the desired data items are far way or unreachable. The latter situation may be caused by network partitioning [40, 85] or client disconnection [11].

The inherent shortcomings of the pull-based, push-based, hybrid and MANET mobile systems lead to a result that systems adopting these architectures alone would not be as appropriate in most real commercial settings. In reality, long access latency or access failure could possibly cause the abortion of valuable transactions or the suspension of critical activities, so that it is likely to reduce user satisfaction and loyalty, and potentially bring damages to the enterprises involved.

The drawbacks of the existing mobile data dissemination models and the state-of-the-art P2P communication technologies motivate us to combine the P2P communication technology with the conventional mobile system, wherein the MHs can obtain their desired data items either from the MSS or their peers, to provide a new communication platform for deploying wireless information access applications. For instance, in an exhibition, the MHs crowded near the same booth are probably interested in information related to the booth. In COCA, after some MHs standing near to the booth downloaded the information about it from the server, they can share the information with other peers, so that the late-coming peers need not access the information from the server again. Furthermore, in a conference, the participants staying in the same session room also likely possess common research interests. They may download similar documents of the proceedings or related works from the server. Once an MH downloads a document from the server, all other peers who are inter-

ested in this document can retrieve it from that MH without enlisting the MSS for help.

In this thesis, we focus on designing a cooperative caching system for the MHs, and proposing a family of algorithms for the MH to search their required data items in the cache of their peers and cooperatively manage the cache of the MH itself and its peer.

1.4 Problem Statements

In this thesis, several key research issues on mobile cooperative caching are studied and addressed, as follows:

- **The Combination of P2P Communication Technologies and the Conventional Mobile Environment**

  To combine P2P communication technologies into the conventional mobile environment, a set of new communication protocols has to be developed. There should be a *mobile cooperative caching framework* involving P2P communication technologies tailored for each conventional mobile environment, i.e., pull-based, push-based and hybrid conventional mobile environments, in which the MHs can search their peers' cache for their desired data items and retrieve them from either their peers or the server effectively and efficiently.

- **An Information Filtering Mechanism for the MHs**

  As the MHs lack the knowledge about the cache content of their peers, they have to exhaustively search the peers' cache for their desired data items whenever they encounter local cache misses. To alleviate this problem, the MHs have to exchange a summary of their cache content with their peers. The cache summaries should provide sufficient hints for the MHs to determine whether their

desired data items are cached by their peers based on their local information, so that they can decide whether to search the peers' cache for their desired data items or retrieve them from the server directly.

It is costly to exchange and maintain the cache summary between two MHs, so that the MHs only exchange their cache summaries with their neighboring peers. Furthermore, since neighbor join or departure frequently occurs in a mobile environment, it is also costly to exchange or maintain the cache summary of all neighbors. To address this problem, the MHs should only exchange and maintain the cache summary of their neighboring peers *with similar mobility patterns*, so that the number of cache summary retrieval and removal operations can be significantly reduced.

- **Cooperative Cache Management**

  The major issues in cooperative cache management are the proper choice of cache replacement and admission control strategies, so as to increase the data accessibility, not only with respect to individual clients, but also to other peers. Furthermore, client mobility and data access patterns are key factors to system performance and cache management strategies in a mobile environment. To deal with these issues, cooperative cache admission control and cache replacement strategies considering the client mobility and data access patters should be designed for the MHs.

1.5 Contribution of the Thesis

In this thesis, we first propose a cooperative caching system for mobile environments in the pull-based, push-based and hybrid environments. A cache signature scheme using signature techniques for information filtering is next proposed for the MHs adopting

COCA. Also, two group-based COCA schemes with cooperative cache management protocols are proposed and studied on top of COCA system and cache signature scheme. Figure 1.1 provides a big picture of the research in this thesis, exhibiting the relationship between different parts of the layered COCA system, cache signature scheme and two group-based COCA schemes.

1.5.1 COCA: A Model for Mobile Cooperative Caching

A mobile cooperative caching model, namely COCA, is proposed for three different kinds of mobile data dissemination models: pull-based, push-based and hybrid. For each data dissemination model, a system model and a set of communication protocols are designed. In addition, we extend COCA to multi-hop P2P data searching to further improve system performance.

1.5.2 Signature Techniques for Information Filtering

In COCA, since the MHs do not possess any knowledge about the cache content of their peers, they have to exhaustively search the peers' cache for their desired data items whenever they encounter local cache misses. To alleviate this problem, a cache signature scheme is proposed for the MHs to provide hints for them to determine whether the required data items are cached by their peers based on the local

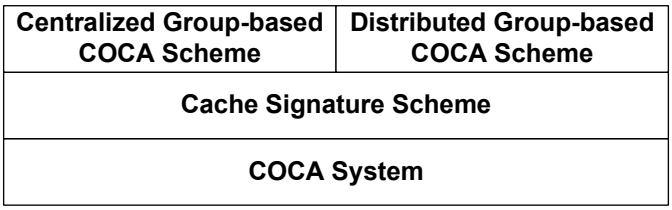


Figure 1.1: The relationship between different parts of this thesis: the COCA system, cache signature scheme and two group-based COCA schemes.

knowledge. Three different signature storage schemes are proposed, with different tradeoff between the storage space and maintenance overhead. In the group-based COCA schemes, the information provided by the cache signatures is also used for a cooperative cache replacement protocol to improve data accessibility.

1.5.3 Group-based Cooperative Cache Management

Two group-based COCA schemes, *centralized* and *distributed* group-based COCA schemes, extending the standard COCA with cache signature scheme are proposed. The group-based COCA schemes cluster the MHs into groups based on their mobility patterns and data affinities. We design two cooperative cache management protocols: *cooperative cache admission control* and *cooperative cache replacement*, for the group members to manage their cached data items cooperatively, in order to improve system performance.

The performance of COCA, COCA with cache signature scheme and the group-based COCA schemes is extensively evaluated through a number of simulated experiments to investigate their effectiveness, robustness, scalability and adaptability.

1.6 Organization of the Thesis

In this chapter, we have briefly described the background of mobile data dissemination models, mobile systems and mobile cooperative caching. We have also discussed the motivation and contribution of this thesis. The rest of this thesis is organized as follows:

- Chapter 2 gives a survey on some important works related to cooperative caching in three different types of environments, including distributed shared-memory systems in wired networks, hierarchical web caching in the Internet,

and mobile environments.

- Chapter 3 presents a cooperative caching scheme, namely COCA, in three different kinds of mobile environments: pull-based, push-based and hybrid. Additionally, COCA with multi-hop P2P data searching is also described in this chapter. We conduct performance studies of COCA based upon simulated experiments.

- Chapter 4 depicts a cache signature scheme for efficient data filtering in COCA. After describing the background of signature techniques and a proactive generation mechanism of the cache signature scheme, three signature storage schemes and their exchange protocols will be proposed. We then evaluate the performance of COCA with cache signature scheme by conducting a number of simulated experiments.

- Chapter 5 presents two group-based COCA schemes: *centralized* and *distributed*, namely CGCoca and DGCoca respectively. After defining the family of algorithms in CGCoca and DGCoca, two cooperative cache management protocols, *cooperative cache admission control* and *cooperative cache replacement* will then be proposed. The performance of the group-based COCA schemes is extensively evaluated through a number of simulated experiments.

- Chapter 6 offers concluding remarks on this thesis, and outlines some potential future works for COCA.

Chapter 2

Related Works

In this chapter, we present some important works related to cooperative caching in three different types of environments, including distributed shared-memory systems in wired networks, hierarchical web caching in the Internet, and mobile environments.

2.1 Cooperative Caching in Distributed Shared-Memory Systems in Wired Network

We first outline some important works on cooperative caching for distributed shared-memory systems in wired networks (e.g., LANs or local area networks), including N-Chance Forwarding [28], GMS (Global Memory Service) [31] and Hint-based approach [74]. Generally speaking, they can be divided into two categories: *tightly* and *loosely coordinated* cooperative caching algorithms. The N-Chance Forwarding and GMS work are concerned with tightly coordinated systems, whereas hint-based cooperative caching is related to a loosely coordinated system.

2.1.1 N-Chance Forwarding

N-Chance Forwarding [28] is a centralized, fact-based cooperating caching algorithm that all clients execute to cooperate in order to preferentially cache *singlets*, i.e., the blocks that are the only copy in the global cache. For each singlet, it is associated with a *recirculation count* that is set to N when it is first forwarded. To replace a singlet, the singlet is randomly forwarded to another peer rather than simply evicting it from the cache to improve data accessibility in the system. When a singlet becomes the least valuable data in the LRU (Least Recently Used) list, if the value of the recirculation count is larger than zero, the client decrements the value of the recirculation count and forwards it to a randomly chosen peer. Otherwise, the client simply discards the singlet so that the unused blocks are eventually discarded from the cache. If a singlet is accessed by any client again, the recirculation count is reset to N. Additionally, this algorithm dynamically adjusts the client cache's allocation between local data and global data based on the client activity.

In [28], a centralized file server is dedicated to keep track of all block location information and singlet status. When a client has to discard a block, if the block is not associated with a recirculation count, the client needs to check its singlet status with the file server. Also, when a client forwards a singlet to another peer, the client has to inform the server about the location change of the forwarded singlet.

2.1.2 GMS

GMS implemented by Feeley et al. [31] is another centralized, fact-based cooperative caching algorithm approximating a cluster-wide global LRU replacement strategy in a distributed shared-memory system. In comparison with N-Chance Forwarding, GMS is implemented with a more strictly coordinated replacement mechanism. Time is divided into epochs. At the start of each epoch, a designated workstation, namely

initiator, is selected to collect global information about the ages of cached pages on all cluster nodes. Then, the initiator assigns a weight to each workstation by calculating the number of the oldest pages stored by that workstation out of a maximum number of cluster replacements for the oldest pages in the epoch, and minimum age of the pages that will be replaced in the epoch. The initiator disseminates the calculated weights and minimum age to all workstations in the cluster. The workstations determine the action to be taken when they encounter a page fault based on the given information. The experiment results show that GMS gives better performance than N-Chance Forwarding, especially in an environment with a skewed distribution of idle memory in the cluster.

Veolker et al. [84] extended GMS to consider not only age information, but also client load information. In that variant, three types of nodes are defined: *idle*, *local* and *global* nodes. The memory requirements of the jobs at local nodes can be satisfied by their local memory. For the global nodes, the memory requirements of their jobs are beyond their local memory, i.e., they need to enlist help from other nodes with global memory. Lastly, there is no job running on idle nodes. The local and idle nodes may provide memory space for the global nodes. A global page replacement algorithm is proposed to balance the global memory traffic load across the idle nodes, bound the global memory traffic load on local nodes and hinder the global nodes from handling any global memory requests.

Prefetching Global Memory System (PGMS) [83] further extends GMS by adopting cooperative prefetching techniques to reduce delay on fetching missing pages. There are two levels of prefetching: *local* (disk-to-local and global-to-local) and *global* (disk-to-global). For local prefetching, a page is prefetched from the network, if it is in the global memory; otherwise, the desired page is prefetched from the disk. For global prefetching, a page is prefetched from the disk to the global memory by another peer.

Then, the page will be prefetched from the global memory of the peer to the local cache of the requesting workstation. The experiment results show that PGMS gives better performance than the extended GMS [84].

2.1.3 Hint-based Cooperative Caching

Hint-based cooperative caching scheme [74] is a loosely coordinated cooperating caching scheme applied in a file system that allows clients to make decisions based on local state or hints rather than global state or facts to reduce the coordination overheads, thereby making caching more effective. In the hint-based system, a manager is dedicated for keeping track of the location information of only one copy of each block, namely *master copy*. The master copy of a block is the first copy obtained from the server as no client is caching the block. When a client opens a file, it consults the manager to get a set of hints containing the probable location of the master copy of all required blocks of the file. To maintain the accuracy of the hints, the manager requests the location information of the master copy of every block of a file from the last client that has opened it.

A *best-guess replacement* algorithm is proposed to approximate the global LRU replacement strategy. Each client maintains an oldest-block list that is sorted in descending order according to the client's best guess on the age of the oldest block cached by other clients. When a client needs to replace a block in the cache, the client determines whether to forward the victim to another client. If the victim is a master copy, the client forwards it to the client in the head of the oldest-block list; otherwise, the victim is simply evicted from the cache. To maintain an accurate oldest-block list, the clients exchange the age of their current oldest block after forwarding a block. The clients update their lists based on the received age information.

The experiment results depict that hint-based cooperative caching scheme per-

forms better than N-Chance Forwarding, and gives comparable performance to GMS, but hint-based cooperative caching can significantly reduce the amount of messages for block access and cooperative cache replacement.

2.2 Cooperative Caching in Hierarchical Web Caching

Hierarchical web caching is proposed as a strategy adopted in the Harvest project [20] for Internet information access. The basic idea of hierarchical web caching is that when a proxy cache encounters a local cache miss, it first requests other nearby proxies before obtaining the desired data object from the original server. The nearby proxies and the requesting cache actually form a certain hierarchical structure. A parent proxy is one level up in the hierarchical structure, whereas a sibling proxy is on the same level. If no sibling proxy returns the desired data object, the requesting proxy forwards the request to its parent at the upper level of the hierarchy. This lookup process will be iterated until the desired data object is found. If the root of the hierarchy is reached and no proxy holds the desired data object, the object will be retrieved from the object's original server. The proxies residing in the hierarchy involved in returning the desired data object to the requesting proxy also cache a copy of it. The experiment results show that hierarchical web caching can improve average response time and reduce server load effectively. A popular open source web proxy, called Squid [1, 86], is also derived from the Harvest project.

A simple communication protocol, called Internet Cache Protocol (ICP) [86, 87, 88], that is an application layer protocol running on top of User Datagram Protocol/Internet Protocol (UDP/IP), is developed to search and retrieve desired data objects from the proxies in the hierarchical web caching. When a proxy encounters a

local cache miss, it sends an ICP query to its sibling proxies to request for the desired data object. The proxies reply the requesting cache with ICP replies indicating either a *hit* or *miss*. The sibling proxy that caches the data object turns in the data object to the requesting proxy. If no sibling proxy stores the desired data object, i.e., there is a miss in the same hierarchy level, the requesting proxy has to forward the request to its parent.

Since cooperative caching in hierarchical web caching is extensively studied, there are many relevant research works. In this section, we merely present some of them that focus on hash-based routing protocol and proxy cooperation.

2.2.1 Hash Routing Protocol

ICP is robust, as there may be several copies of a data item in the hierarchical web caching. When a web proxy goes down, the data items cached by it are likely to be still accessible in other web proxies. This nice characteristic does provide fault tolerance for the system. However, this characteristic probably reduces the effective aggregate cache size of the entire hierarchy. Additionally, the searching overhead in ICP is expensive, as the client proxy has to broadcast an ICP request to all other proxies to seek a desired data item.

To increase the effective cache size and reduce the searching overhead, an alternative routing protocol, termed hash routing protocol, is proposed. Highest random weight [81] (HRW), is a hash routing protocol, in which a randomized weight function is defined by including the data item name and the Internet Protocol (IP) address of a server. When a client encounters a local cache miss, it calculates a score for each available server with the name of the desired data item by the weight function. The client sends the request for the desired data item to the server with the highest (or lowest) score. HRW possesses several properties: load balancing, low disrup-

tion for server admission and removal, and low overhead of the weight calculation. The simulated experiment results show that HRW outperforms other mapping methods, including round-robin, random allocation and least-loaded allocation schemes, in terms of response time, hit ratio and disruption overhead.

In a realistic web caching hierarchy, the proxies may possess different capacity, such as cache capacity and processing power. An extended hash routing protocol [71] is proposed to take this heterogeneous environment into consideration. The idea of the protocol is to assign more data items to the proxies with higher capacity that can be done by multiplying an additional factor to the calculated score. The additional factor is obtained from a recursive calculation based on the target access probability of each proxy. The proposed hash routing protocol is then implemented in Cache Array Routing Protocol (CARP) [82].

2.2.2 Summary Cache and Cache Digests

Although ICP is simple and easy to implement, it is not scalable. Li et al. [30] demonstrate that the communication and computation processing overhead increase quadratically with increasing number of proxies in a cache hierarchy. Also, Rousskov and Wessels [72] mention that the amount of network traffic is proportional to the product of the number of proxies in a hierarchy and the number of Hypertext Transfer Protocol (HTTP) requests. A new protocol, namely *summary cache* [30], is proposed to reduce network traffic among web proxies. The cache content of a proxy is summarized by a signature. By exchanging the signature among all participating proxies, each proxy can determine whether the required data items are likely cached by other proxies. When a proxy experiences a local cache miss, and the received signatures indicate that some of other proxies are likely caching the required data items, the proxy sends a request to the relevant proxies. Otherwise, the proxy forwards the

request to the original server. A similar protocol is proposed by Rousskov and Wessels [72], called *cache digests*. Both the cache digest and summary cache protocols adopt standard 128-bit MD5 hash function [70], and store the hash key in a bloom filter [15]. The major differences between summary cache and cache digests are as follows:

1. Cache digest protocol adopts a *pull* technique for cache digest exchange, whereas a *push* technique is used by summary cache protocol.

2. Summary cache protocol uses a counter vector structure to produce cache signatures, in order to save computation processing overhead, but cache digest protocol does not use any structure to support the generation process. The details of the counter vector structure will be presented in Chapter 4.

2.2.3 Expiration-Age based Scheme

In cooperative caching, data placement strategy plays one of the key roles in system performance. Web proxies have to make decisions on data placement when they are involved in data access transaction. An Expiration-Age (EA) based scheme [69] is proposed for document placement for cooperative web caching to exploit the relationship between client disk contention and expiration age of cache. The document placement decisions are based on the average EA of the most recently evicted cache group. When a proxy encounters a local cache miss, it sends a query to its all siblings and parents via ICP. If any one of them can turn in the required document to the requesting proxy, they exchange their own average EA with each other. If the average EA of the requesting proxy is larger than that of the document source proxy, the requesting proxy caches the document in its local cache; otherwise, it does not store it locally. Likewise, the source proxy compares its own average EA against that

of the requesting proxy. If the average EA of the source proxy is larger, it promotes the document to the head of the LRU list to extend its time-to-live; otherwise, no promotion is performed. The EA scheme can reduce the number of document replicas across the shared web cache proxies and increase the life of cached documents. There is no extra connection setup in EA scheme, as the EA information is piggybacked on the HTTP messages.

2.3 Cooperative Caching in MANETs

Recently, cooperative caching schemes in mobile environments have been drawing increasing attention. During the preceding years, several cooperative caching schemes were proposed for mobile environments. These works can be divided into two major categories: *cooperative data dissemination* and *cooperative cache management*. The work of cooperative data dissemination mainly focuses on how to search for desired data items, and how to forward the data items from the source MH or the MSS to the requesting MHs. The work pertaining to cooperative cache management focuses on how mobile clients can cooperatively manage their cache as a global cache or aggregate cache to improve system performance along such design dimensions as data replication, cache invalidation, cache replacement and cache admission control. In this thesis, our primary focuses are upon designing a cooperative data dissemination protocol for the COCA system and proposing cooperative cache management protocols for the group-based COCA schemes.

2.3.1 Cooperative Data Dissemination

Research in [73] proposed an intuitive cooperative caching architecture for a MANET environment on top of Zone Routing Protocol (ZRP) [38, 39]. ZRP is an ad-hoc

routing protocol in which the MHs apply a proactive routing scheme within their own zone that is defined by a zone radius parameter (in hops), and apply a reactive routing scheme outside their zone. If an MH can directly connect to a MSS with a single hop, it would directly obtain the required data items from the MSS; otherwise, the MH has to enlist its peers at a distance less than the MSS for help to turn in the required data items. If no such peer caches the data items, the peers route the request to the nearest MSS. A local cache replacement strategy is also proposed for the MH based the access probability and time-to-live of the cached data items, and the estimated energy cost of retrieving them.

A similar cooperative caching architecture is proposed in [55] that is designed to support continuous media access in MANETs. In [55], two data location schemes, namely *cache-state* and *reactive*, are proposed for the MHs to determine the nearest source that can be the cache of their peers or the original servers to retrieve the required multimedia objects. Cache-state is a proactive scheme, whereas reactive is an on-demand scheme. The simulated experiment results show that reactive outperforms cache-state in terms of network traffic, quality of service (QoS) and access latency.

In [66], a cooperative caching scheme, called 7DS (Seven Degrees of Separation), is used as a complement to the infrastructure support with power conservation. When an MH fails to connect to the Internet to retrieve the desired data item, it would attempt to search for it from its neighboring 7DS peers. The proposed power conservation scheme adjusts the MHs' activity of their participation in the cooperative caching scheme based on their battery levels. The client battery power is divided into three levels: *high* (the default value in the simulation model is above 75%), *medium* (50% to 75%) and *low* (below 50%). The MHs with high and medium battery power level take part in active and passive participation respectively. For the those with low power level, they stop participating in the 7DS to conserve power.

Shen et al. [77] propose another data dissemination scheme with power conservation, called *energy-efficient cooperative caching with optimal radius* (ECOR), in a cooperative caching environment. In ECOR, an optimal radius (in hops) is estimated by an analytical model that considers the MH's location, data access probability and network density for each data item. The MHs exchange the cache content and the optimal radius of each cached data item among themselves. When an MH encounters a local cache miss, if it finds that any peers cache the desired data item and the distance between them is within the optimal radius based on its local state, the MH sends a request message to the peer that is the closest to the holder of the desired data item. Otherwise, the MH obtains the data items from the MSS.

2.3.2 Cooperative Cache Management

In cooperative cache management, all related works can be further divided into three sub-categories: *cooperative data allocation*, *cooperative cache invalidation*, and *cooperative cache admission and replacement*.

2.3.2.1 Cooperative Data Allocation

In [40, 41, 42, 43], replication techniques are adopted in cooperative caching schemes to improve data accessibility, in order to alleviate the network partitioning problem. Hara [40] proposes three replica allocation schemes: *SAF* (Static Access Frequency), *DAFN* (Dynamic Access Frequency and Neighborhood) and *DCG* (Dynamic Connectivity based Group). The MHs applying SAF only consider their own individual access probability to each data item. DAFN extends SAF to take the access probability to each data item of the MHs' connected neighborhoods into account. Finally, DCG groups the MHs with highly stable connection together. A group of MHs pos-

sesses a high connection stability, as they form a biconnected component[1] in the network. DCG considers the access probability to each data item of all MHs in the same group. The simulation results show that DCG gives the highest data accessibility, but it incurs much more network traffic than the other two schemes. Thus, DCG can be considered as a scheme that trades network traffic for data accessibility.

The proposed replica allocation schemes are then adopted to a mobile broadcast environment [41]. In [41], the replica allocation schemes are based on not only the access probability, but also the latency on accessing data items from peers and the broadcast channel. In [42], the proposed replica allocation schemes are further extended to consider periodic data update by allowing the extended allocation schemes to consider the remaining time until each data item is updated. In addition to the access probability, the stability of radio link is also taken into account in [43]. The stability of a radio link is defined as the remaining time period that two MHs will still be connected to each other. The longer the time period indicates the higher the stability of a ratio link.

Huang et al. [48] propose another distributed data replica allocation scheme in MANETs, namely DRAM, to improve data accessibility and reduce network traffic pretending to the replication mechanism. DRAM extends E-DCG [42] to consider group mobility for data replica allocation. The authors in [48] assume that some MHs tend to roam together and they share a common access range. To discover the group mobility among MHs, a distributed clustering algorithm is adopted to cluster several MHs who possess similar mobility pattern into a group. The clustering algorithm is executed periodically to adapt to the changes in network topology. Then, the data replicas are allocated to each group member based on their access frequencies of the data items, and the remaining time until the next update on them. The experiment

[1] In graph theory, a biconnected component [8] is a maximal subset of edges of a connected graph such that the corresponding induced subgraph cannot be disconnected by deleting any one vertex.

results show that DRAM performs better than E-DCG in terms of data accessibility and network traffic.

2.3.2.2 Cooperative Cache Invalidation

Two cache invalidation schemes are proposed in [45], namely *update broadcast* and *connection rebroadcast*. The former one is a straightforward, flooding-based scheme. An MH that caches an original copy of a data item broadcasts an invalidation report to other peers, when that MH updates the data item. The latter one can be referred to as a cooperative cache invalidation scheme. When two MHs are newly connected to each other, they broadcast their collected cache invalidation information. The newly connected MHs and other peers receiving their broadcast information update their own previously received cache invalidation information to identify any obsolete data items in their cache. The experiment results show that connection rebroadcast scheme reduces the number of accesses to invalid cached data items, but it incurs higher network traffic than update broadcast scheme.

2.3.2.3 Cooperative Cache Admission Control and Replacement

Lim et al. [63] propose a cooperative caching scheme for Internet-based MANETs, namely IMANET. In IMANET, a simple, flooding-based searching scheme is proposed to lookup desired data items in the network. That cooperative caching scheme also provides two functions: cache admission control and cache replacement. For the cache admission control, an MH determines whether to cache a data item based on the distance between itself and the data source that can be either other peers caching the data item or the MSS. For the cache replacement policy, called *time and distance sensitive* (TDS), a victim data item is selected to be evicted from the cache by an MH based on two factors: the distance between itself and other peers caching the victim

or the MSS, and the freshness of the distance information. The distance information is updated when the corresponding data item is accessed by other MHs. Since the network topology changes frequently, the distance information could become obsolete, as it has not been updated for a long time.

Yin and Gao [91] propose three other cooperative caching schemes, called *CacheData*, *CachePath* and *HybridCache*. The idea of CacheData is that an MH caches a passing-by data item, if the data item is popular and a condition that all requests for the data items are not originated by the same MH is satisfied. For CachePath, the MH caches path information, i.e., a data item is likely to be cached by which MHs, of the passing-by data item instead of the data item. To conserve cache space, an MH does not cache path information of all passing-by data items. It only caches the path information of a data item, if it is closer to the requesting MH than the MSS. HybridCache is a hybrid scheme which combines both CacheData and CachePath. An MH either applies CacheData or CachePath based on three factors: the size of a data item, the time-to-live of a data item and the distance between the MH's distance to the data holder and the distance to the MSS. If the size of a data item is small or its time-to-live is short, CacheData is adopted. In case of the distance between the MH's distance to the data holder and the distance to the MSS is large, CachePath is chosen. The simulation results exhibit that HybridCache outperforms CacheData and CachePath.

Chapter 3

COCA

3.1 Introduction

In this chapter, we design a mobile cooperative caching system, called COCA, for the MHs in three popular data dissemination models in mobile environments: pull-based, push-based and hybrid models. For each data dissemination model, a relevant COCA model and a set of communication protocols are presented. A multi-hop P2P data searching protocol is also proposed for COCA. The performance of COCA in the three data dissemination models is extensively evaluated through a number of simulated experiments. The result shows that COCA effectively improves system performance in terms of access latency and server workload in mobile environments. COCA generally incurs higher power consumption in a pull-based environment, but it can effectively reduce power consumption in push-based and hybrid environments.

3.2 Assumptions in COCA

In COCA, we assume that each MH has a unique identifier. Let the set of MHs be $\mathcal{M}$=$\{m_1, m_2, ...,m_{NumClient}\}$, where $NumClient$ is the total number of MHs in the

system. Furthermore, each MH is equipped with two wireless network interface cards (NICs), in which one is dedicated to communicate with MSS, while the other one is devoted to communicate with other MHs. This multi-NIC technique is also proposed to be adopted in IP multi-homing for stream control transmission protocol (SCTP), in which a machine is installed with multiple NICs, and each assigned a different IP address [79]. There are two P2P communication paradigms: point-to-point and broadcast. In P2P point-to-point communication, there is only one destination MH for the message being sent from the source MH. In P2P broadcast communication, all MHs residing in the transmission range of the source MH receive the broadcast message. In addition, a message sent by a source MH is assumed to be correctly received by the destination MHs within a finite time. For simplicity, we further assume that there is no update of data items.

3.3 System Model

We first consider a pull-based mobile environment with COCA. Then, COCA is extended to two other models, push-based and hybrid data dissemination models. Before describing the system model, we first discuss the four possible outcomes of each client request.

1. **Local Cache Hit (LCH).** If the required data item is found in the MH's local cache, it constitutes a local cache hit; otherwise, it is a local cache miss.

2. **Global Cache Hit (GCH).** When the required data item is not cached, the MH attempts to retrieve it from its peers. If some of the peers can turn in the required data item, that constitutes a global cache hit.

3. **Cache Miss (or Server Request).** If the MH fails to achieve neither a local

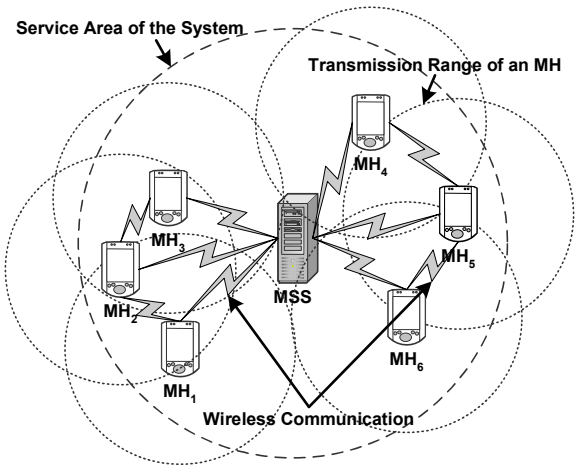


Figure 3.1: System architecture of COCA.

cache hit nor a global cache hit, it encounters a cache miss, and has to make a connection to the MSS to obtain the required data item.

4. **Access Failure.** If the MH encounters a cache miss, and fails to access the required data item from the MSS, as it is residing outside of the service area or the MSS is being down or overloaded, it constitutes an access failure.

COCA is based on the system architecture, as depicted in Figure 3.1. Each MH and its peers work together to share their cached data items cooperatively via P2P communication. For instance, when an MH, MH_2, encounters a local cache miss, it broadcasts a request to its peers, MH_1 and MH_3, as shown in Figure 3.1. If any peers turn in the required data item, a *global cache hit* is recorded; otherwise, it is a *global cache miss*, and MH_2 has to enlist the MSS for help.

3.3.1 Multi-hop P2P Data Searching in COCA

In COCA, there are three types of messages: request, reply and retrieve. A request message contains a unique identifier that is formed by a combination of the user identifier ($UserID$), the time stamp when the request is generated ($RequestTS$), and a hop

counter (*HopCount*), which is initially set to *HopDistance*, i.e., <*UserID*, *RequestTS*, *HopCount*>. Each peer only processes a request with the same request identifier once, and it simply drops any duplicate requests. If the MH caches the required data item, it sends a reply message to the requesting MH directly or through multi-hop routing. Otherwise, it decrements the *HopCount*, and if $HopCount > 0$, it propagates the request to its neighborhood. If $HopCount = 0$, the MH simply drops the request. After the requesting MH receives a reply message from its peer, it sends a retrieve message to the peer, and the peer then turns in the required data item to it, either via direct communication or multi-hop routing. In multi-hop P2P data searching, the immediate MHs not only propagate the requests to other peers, but they also have to forward the required data items from the source MH to the requesting MH.

3.3.2 Pull-based Mobile Environment

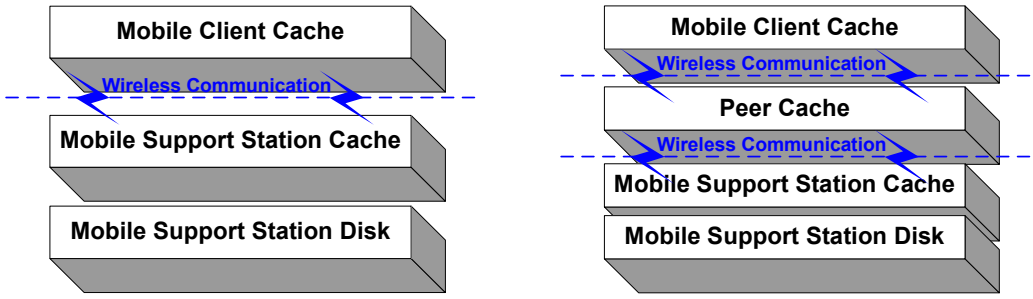
In a conventional pull-based mobile system, the storage hierarchy is generally composed of three layers: Mobile Client Cache, MSS Cache and MSS Disk, as depicted in Figure 3.2(a). When an MH encounters a local cache miss, it sends a request to the MSS. The MSS processes the request and sends the data item back to the requesting MH. If the data item is not located in the MSS cache, the MSS has to grab the data item from the disk or database server. In COCA, a new logical layer is inserted between the Mobile Client Cache layer and MSS Cache layer, called Peer Cache layer [25], as depicted in Figure 3.2(b). When an MH encounters a local cache miss, it first attempts to search for the desired data items in the Peer Cache layer. If no peer caches the data item, it obtains the data item from the MSS.

The communication protocol of COCA in a pull-based mobile environment specifies that an MH should first find the required data item in its local cache for each query. If it encounters a local cache miss, it broadcasts a request message to its peers within

the distance of a predefined maximum number of hops, i.e., *HopDistance*, via P2P broadcast communication. Any peer caching the required data item will send a reply message to the requesting MH via P2P point-to-point communication. When the MH receives some reply messages, it selects the most appropriate peer as a target peer. The selection of the target can be based on different metrics, as follows:

- the peer with the fastest response time that could reduce the access latency;
- the peer with the shortest distance that could reduce the power consumption, as power control mechanism is adopted [5, 51];
- the peer with the highest battery level that could increase the overall battery lifetime;
- the most trustworthy peer that could increase the reliability of the obtained information;
- and, the peer who caches the information with the longest time-to-live that could increase the validity of the obtained information.

The MH next sends a retrieve message to the target peer via P2P point-to-point communication. The peer receiving the retrieve message turns in the required data



(a) Conventional **(b)** COCA

Figure 3.2: Storage hierarchy of pull-based mobile systems.

item to the requesting MH through P2P point-to-point communication. In case of no peer sending reply message back to the requesting MH during a timeout period, it has to obtain the data item from the MSS.

The timeout period is adaptive to the degree of network congestion. Let |request| and |reply| denote the message size of a request and reply message respectively. Initially, the timeout period is set to a default value that is defined as the round-trip time of a P2P transmission scaled up by a congestion factor, φ, i.e., $\frac{(|\text{request}|+|\text{reply}|)}{BW_{P2P}} \times HopDistance \times \varphi$, where BW_{P2P} is the bandwidth of the P2P communication channel and $HopDistance$ is the distance of a predefined maximum number of hops. For each search in the Peer Cache layer, an MH records the time, τ_r', spent on the duration from the time when the MH broadcasts a request message to the time when a reply message is received. Then, the timeout period, τ_r, is set to the average time duration, $\overline{\tau_r'}$, plus another system parameter, φ', times the standard deviation of τ_r', $\sigma_{\tau_r'}$, i.e., τ_r is set to $\overline{\tau_r'} + \varphi'\sigma_{\tau_r'}$.

3.3.3 Push-based Mobile Environment

In a traditional push-based mobile system, the storage hierarchy commonly consists of three layers: Mobile Client Cache, Broadcast Channel and MSS Disk, as shown in Figure 3.3(a). The MSS grabs the data items from the disk or database, and allocates them to the broadcast channel. If an MH encounters a local cache miss, it tunes in to the broadcast channel, and catches the required data item when the data item appears in the broadcast channel. In COCA, we insert the same logical layer, the Peer Cache layer, as in the case of pull-based mobile environment, as a supplementary component of the Broadcast Channel layer, as shown in Figure 3.3(b). When an MH suffers from a local cache miss, the MH tunes in to the broadcast channel; meanwhile, it searches for the required data item in the Peer Cache layer. There is a global cache

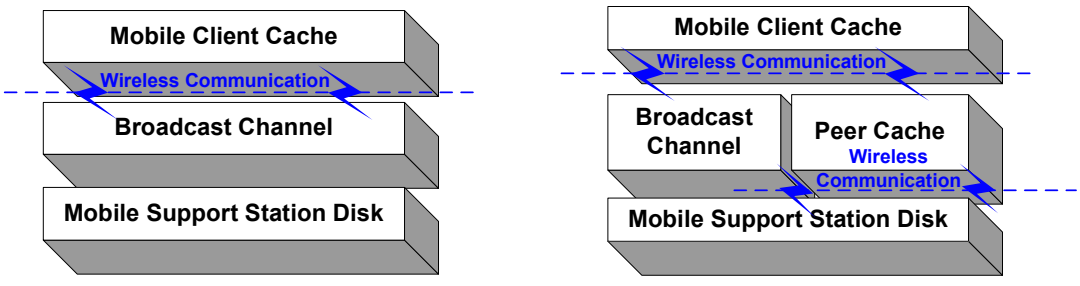
hit, if some peers turn in the required data item to the MH before either the data item appears on the broadcast channel or the timeout period elapses. In case of no peer returning the required data item, i.e., a global cache miss, the MH has to wait for the data item to appear in the broadcast channel, as in the conventional case.

3.3.4 Hybrid Mobile Environment

In a hybrid mobile environment, MHs make use of both point-to-point and broadcast channels to retrieve data items from the MSS. We apply three techniques to the hybrid data dissemination model: bandwidth allocation, data allocation and broadcast channel indexing.

3.3.4.1 Bandwidth Allocation

In COCA, a static channel allocation scheme is used, and the channel allocation is based on a ratio, namely *PushChannel*, which indicates the percentage of channels that are allocated for data broadcast. For instance, if there is a total number of 20 channels and *PushChannel*=30%, then six channels are reserved for data broadcast, while the remaining channels are devoted to point-to-point communication.



(a) Conventional **(b)** COCA

Figure 3.3: Storage hierarchy of push-based mobile systems.

3.3.4.2 Data Allocation

In a hybrid data dissemination system, the MHs can access the hot data items (up to a total size of all disk sizes) via the broadcast channel to improve system scalability. On the other hand, to access the remaining cold data items, the MHs still have to obtain them from the MSS via the point-to-point communication.

The selection of data items to be allocated to the broadcast channel is based on their access probabilities, which can be estimated by observing their access frequencies, f. The MSS records the relative access frequency of each data item to estimate its access probability. For data item, i, f_i is initially set to zero and the last access time, t_i, is set to the time of initialization, and then f_i is updated as it is requested by an MH via the point-to-point communication based on the equation:

$$f_i^{\mathsf{new}} = \omega \times \frac{1}{\mathsf{now} - t_i} + (1 - \omega) \times f_i^{\mathsf{old}}, \tag{3.1}$$

where ω $(0 \leq \omega \leq 1)$ is a parameter to weight the importance of the most recent access. Then, t_i is set to the current time (now). The data allocation is performed at the beginning of each analysis period (*AnalysisPeriod*), based on the weighted access frequencies of the data items, which are approximately proportional to their access probabilities.

Since the access frequency collected by the MSS belongs to "past" information, it can only be used to predict the trend of the access pattern. The hottest data items are likely to be cached by most MHs, so they need not to be broadcast with the highest frequency. Thus, a parameter, namely *Offset*, is used to determine the number of the hottest data items that should be shifted from the disk with the highest spinning speed to the one with the lowest spinning speed [3].

3.3.4.3 Broadcast Channel Indexing

The MSS periodically broadcasts the index of the data items being broadcast in a broadcast cycle to the MHs, so that the MHs are able to determine whether the required data items can be obtained via the broadcast channel based on the index. The index contains the identifier of each data item being broadcast and its broadcast latency, which is defined by subtracting the current broadcast slot from the slot containing the data item. The index is broadcast to the MH every *IndexInterval*. When an MH encounters a local cache miss, it tunes in to the broadcast channel. If the data item appears in the broadcast channel before the index is broadcast, the MH catches the data item. Otherwise, the MH grabs the index and looks up the required data item. If the MH cannot find the identifier of the data item in the index or the latency of accessing it is longer than a latency threshold, *LatencyThreshold*, the MH switches to retrieve the data item from the MSS via the point-to-point channel [3, 47].

3.4 Simulation Model

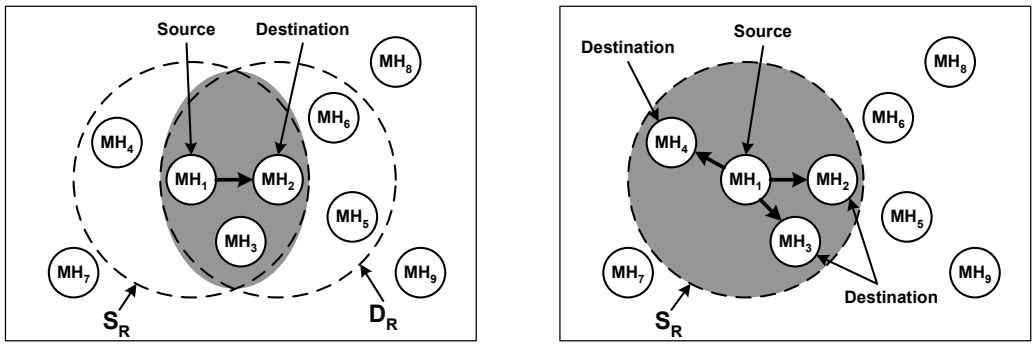
In this section, we present the simulation model that is used to evaluate the performance of COCA in mobile environments. The simulation model is implemented in C++ using CSIM [75], which is a process-oriented discrete-event simulation package. The simulated mobile environment is composed of an MSS and *NumClient* MHs. There are *NumChannel* wireless communication channels between the MSS and the MHs, with a total bandwidth of BW_{Server}. If all the uplink channels are busy, the MH has to wait until one of them is available for transmission. Also, there are *NumP2PChannel* half-duplex wireless channels for an MH to communicate with its peers with a total bandwidth of BW_{P2P} and with a transmission range of $TranRange$. When an MH has to send a message to another MH, it has to wait if the requested

channels are busy.

3.4.1 Power Consumption Model

For the P2P communication, all MHs are assumed to be equipped with the same type of wireless NICs with an omnidirectional antenna so that all MHs within the transmission range of a transmitting MH can receive its transmission. Furthermore, the wireless NIC of the non-destination MH is operated in an idle mode during the transmission.

The communication power consumption measurement is based on [32] which uses linear formulas to measure the power consumption of the source MH, S, the destination MH, D and other *remaining* MHs residing in the transmission range of the source MH, S_R and the destination MH, D_R, and both the source and destination MHs, SD_R, as depicted in Figure 3.4(a), in a MANET. The power consumption of P2P point-to-point communication is measured by Equation 3.2,



(a) P2P point-to-point communication **(b)** P2P broadcast communication

Figure 3.4: Power consumption measurement models for P2P communication.

$$P_{p2p} = \begin{cases} (v_{send} \times |msg|) + f_{send}, & \text{for } MH = S \\ (v_{recv} \times |msg|) + f_{recv}, & \text{for } MH = D \\ (v_{sd\_disc} \times |msg|) + f_{sd\_disc}, & \text{for } MH \in S_R \wedge MH \in D_R \\ (v_{s\_disc} \times |msg|) + f_{s\_disc}, & \text{for } MH \in S_R \wedge MH \notin D_R \\ (v_{d\_disc} \times |msg|) + f_{d\_disc}, & \text{for } MH \notin S_R \wedge MH \in D_R \end{cases} \quad (3.2)$$

where f is the fixed setup cost for a transmission, and v is the variable power consumption based on the size of a message msg in bytes ($|msg|$).

For P2P broadcast communication, the power consumption of source MH, S, and other MHs residing in S_R, as depicted in Figure 3.4(b), can be measured by Equation 3.3.

$$P_{bc} = \begin{cases} (v_{bsend} \times |msg|) + f_{bsend}, & \text{for } MH = S \\ (v_{brecv} \times |msg|) + f_{brecv}, & \text{for } MH \in S_R \end{cases} \quad (3.3)$$

Although the proposed power consumption model in [32] is based on an ad-hoc mode, it is also used to approximate the power consumption of the point-to-point communication between the MH and MSS. When an MH communicates with the MSS, we assume that the surrounding MHs are not affected by the transmission. Thus, no power is consumed by their wireless NICs. Tables 3.1 and 3.2 show the

Table 3.1: Parameter settings for the power consumption model in P2P point-to-point communication.

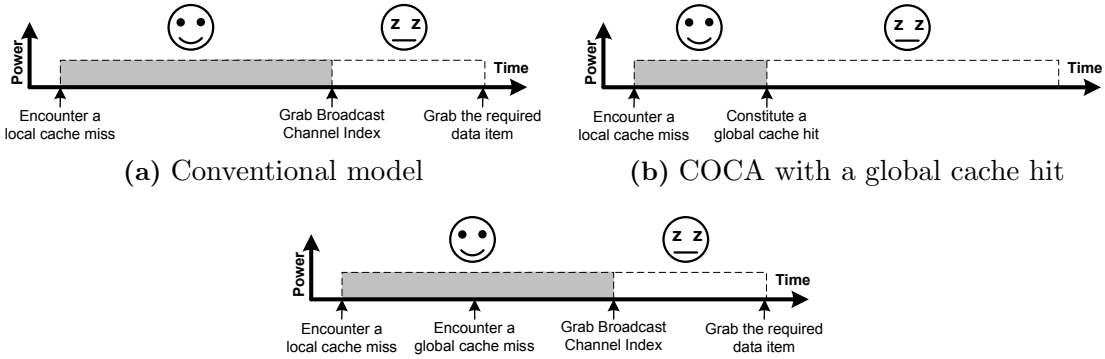
Conditions	$\mu W \cdot$ **s/byte**	$\mu W \cdot s$
$MH = S$	$v_{send} = 1.9$	$f_{send} = 454$
$MH = D$	$v_{recv} = 0.5$	$f_{recv} = 356$
$MH \in S_R \wedge MH \in D_R$	$v_{sd\_disc} = 0$	$f_{sd\_disc} = 70$
$MH \in S_R \wedge MH \notin D_R$	$v_{s\_disc} = 0$	$f_{s\_disc} = 24$
$MH \notin S_R \wedge MH \in D_R$	$v_{d\_disc} = 0$	$f_{d\_disc} = 56$

Table 3.2: Parameter settings for the power consumption model in P2P broadcast communication.

Conditions	$\mu W \cdot$ **s/byte**	$\mu W \cdot s$
$MH = S$	$v_{bsend} = 1.9$	$f_{bsend} = 266$
$MH \in S_R$	$v_{brecv} = 0.5$	$f_{brecv} = 56$

parameter settings for MHs playing different roles, as used in power consumption measurement for P2P point-to-point and broadcast communication respectively [32].

The power consumption measurement model of a broadcast environment is different from that of a pull-based environment, since there is additional power consumption for the MHs listening to the broadcast channel until they obtain the required data items or the broadcast channel index. In the latter case, the MHs can determine when the required data items appear in the broadcast channel by looking up the index. The MHs can keep themselves in a doze mode for a certain period. Then, they wake up and catch the required data items, as shown in Figure 3.5(a). In COCA, there is a chance for MHs to reduce power consumption, when some of their peers can turn in the required data items before the data items or the index appear in the broadcast channel, as depicted in Figure 3.5(b). Figure 3.5(c) shows the situation



(a) Conventional model

(b) COCA with a global cache hit

(c) COCA with a global cache miss

Figure 3.5: Power consumption measurement in a broadcast environment.

when the MHs encounter global cache misses.

3.4.2 Mobility Model

We consider an individual random walk model that is based on "random waypoint" model [17, 18]. At the beginning, the MHs are randomly distributed in $1,000 \times 1,000$ meters ($Area$) space, which constitutes the service area of the MSS. Each MH randomly chooses its own destination in $Area$ with a randomly determined speed s from a uniform distribution $U(v_{min}, v_{max})$. It then travels with the constant speed s. When it reaches the destination, it comes to a standstill for a constant time ($PauseTime$) to determine its next destination. It then moves towards its new destination with another speed s' from the uniform distribution $U(v_{min}, v_{max})$. All MHs with individual random walk pattern repeat this movement behavior during the simulation.

3.4.3 Data Access Pattern

The MHs generate accesses to the data items following a Zipf distribution [92] with a skewness parameter θ. If θ is set to zero, MHs access the data items uniformly. By increasing θ, we are able to model a more skewed access pattern to the data items. The access probability p_i of a data item i is defined by: $p_i = \frac{1}{i^\theta \sum_{j=1}^{NumData} \frac{1}{j^\theta}}$, where $NumData$ is the total number of data items in the system. The time interval between two consecutive accesses generated by an MH follows an exponential distribution with a mean $AccessInterval$.

For the individual data access patterns, we consider two types of access patterns, as follows:

1. Random access range - the access range of every MH is randomly selected.

2. Common hot spot - for all MHs, a certain percentage of the hot spots in their ac-

cess ranges are common, and the remaining access ranges are randomly selected for each MH.

The definition of access density in this work is the same as that defined in [58]. In [58], user access density is defined as a ratio of the total number of requests to the number of distinct data items during a measurement period of time. To vary the access density, we maintain the same number of requests, and change the number of distinct data items, i.e., the access range, being accessed by the MHs. To increase the access density, the number of distinct data items should be decreased. On the other hand, to create a lower access density, we should increase the number of distinct data items being accessed by the MHs.

3.4.4 Server Model

There is a single MSS in the simulated mobile environment. A database in the MSS contains *NumData* equal-sized (*DataSize*) data items. It receives and processes the requests sent by the MHs residing in *Area* with a first-come-first-serve policy. An infinite queue is used to buffer the outstanding requests from the MHs when the MSS is busy.

3.4.5 Network Model

In the simulation, neighbor discovery is based on a neighbor discovery protocol (NDP) [39, 61], which is a protocol that maintains the neighboring connectivity through a periodic beacon of "hello" control message that only contains the source address. Each MH broadcasts "hello" message to its neighboring peers every time period, T_{beacon}. The reception of "hello" beacon indicates the existence of a direct connection to a neighboring MH. On the other hand, if the MH has not received

"hello" beacon from a neighboring MH for a period of time, τ_b, it indicates a disconnection or link failure between the neighboring peer. Algorithm 1 is executed when the MH receives a beacon message from its peers, and Algorithm 2 is a continuous algorithm to detect any link failure between the MH and its neighboring peers. We further assume that the "hello" control message has the highest transmission priority and can be received by the peers within a finite time. Thus, the MHs can precisely detect a new link to a newly admitted neighbor and a link failure with a leaving MH.

Algorithm 1 Neighbor Discovery Protocol (NDP) at m_i - beacon reception

```
1: procedure OnReceiveBeacon(Neighbor 𝒩, MH m_j)
2: // 𝒩 is a set of identifiers of m_i's neighboring peers
3: // m_i receives a "hello" beacon message from m_j
4: last_beacon_ts_j = now();
5: // now() returns the current time
6: if m_j ∉ 𝒩 then
7:    𝒩 ← 𝒩 ∪ {m_j};
8: end if
```

Algorithm 2 Neighbor Discovery Protocol (NDP) at m_i - link failure detection

```
1: procedure LinkFailureDetector(Neighbor 𝒩)
2: for all m_k ∈ 𝒩 do
3:    if now() - last_beacon_ts_k > τ_b then
4:       𝒩 ← 𝒩 − {m_k};
5:    end if
6: end for
```

3.5 Simulation Results

In the simulated experiments, we compare the performance of COCA (denoted as CC) with a conventional caching scheme that does not involve any cooperation among MHs (denoted as non-COCA or NC). The non-COCA and COCA schemes all adopt LRU cache replacement policy. We also consider COCA with two-hop communication (denoted as CC-2H). In CC-2H, the MHs can search the required data items in their

peers within a distance of two hops, i.e., $HopDistance$ is equal to two. We study NC, CC and CC-2H in pull-based, push-based and hybrid environments. For the push-based environment, two broadcast scheduling algorithms: flat disk (denoted as FD) and broadcast disk [2] (denoted as BD), are considered to provide extensive evaluation. In addition, we also study COCA with multi-hop P2P data searching (denoted as CC-MH) in comparison with non-COCA and standard COCA in the pull-based, pure push-based and hybrid environments. All simulation results are recorded only after the system reaches a stable state, in which all MHs' cache are fully occupied, in order to avoid a transient effect. A simulated experiment ends when each MH generates over 2000 requests after the warm up period. Table 3.3 shows the simulation parameters and their default settings used in the simulated experiments.

We conduct the experiments by varying several parameters: cache size, data item size, access patterns, client disconnection probability, mobility speed, number of MHs and hop distance of multi-hop data searching in the system. The client disconnection probability, P_{disc}, is defined as the probability of an MH disconnecting from the network after completing a request for a period of time that follows an exponential distribution with a mean $DisconnectTime$. The MH does not disconnect itself from the network when it is accessing its desired data items. The MHs cannot communicate with disconnected MHs. The performance metrics include access latency, server request ratio, power consumption on communication, LCH ratio and GCH ratio. The access latency is defined as the sum of the transmission time and the time spent on waiting for requested communication channels, if they are busy. Other performance metrics are determined by following equations:

- server request ratio = $\frac{1}{NumClient}\left(\sum_{i=1}^{NumClient}\frac{NumDataItemsFromServer}{TotalNumRequests}\right)$;
- LCH ratio = $\frac{1}{NumClient}\left(\sum_{i=1}^{NumClient}\frac{NumDataItemsFromLocalCache}{TotalNumRequests}\right)$;
- GCH ratio = $\frac{1}{NumClient}\left(\sum_{i=1}^{NumClient}\frac{NumDataItemsFromPeers}{TotalNumRequests}\right)$;

- power consumption = $\frac{1}{NumClient}\left(\sum_{i=1}^{NumClient}\frac{TotalPowerConsumption}{TotalNumRequests}\right)$.

Table 3.3: **Simulation parameters and default settings.**

Parameter	**Description**	**Default Value**
$NumClient$	No. of MHs in the system	100
$NumChannel$	No. of wireless communication channels between MSS and MHs	20
$NumDisk$	No. of broadcast disks in a broadcast environment	3
$DiskFrequency_{push}$	Disk relative frequency for broadcast disks in a push-based environment	3:2:1
$BDiskSize_{push}$	Disk size for broadcast disks in a push-based environment	1000:3000:6000
$DiskFrequency_{hybrid}$	Disk relative frequency for broadcast disks in a hybrid environment	3:2:1
$BDiskSize_{hybrid}$	Disk size for broadcast disks in a hybrid environment	300:1200:1500
$NumP2PChannel$	No. of half-duplex P2P wireless communication channels	1
$AnalysisPeriod$	Time interval of executing the data allocation algorithm	60 s
$Offset$	Shift offset used in the data allocation algorithm	$CacheSize$
$IndexInterval$	Time interval of broadcasting the broadcast channel index	300 data items
$LatencyThreshold$	Latency threshold used in a hybrid environment	900 data items
BW_{Server}	Total bandwidth of wireless communication channels between MSS and MHs	Downlink 10,000 Kbits/s; Uplink 100 Kbits/s
BW_{P2P}	Total bandwidth of P2P wireless communication channels	2,000 Kbits/s

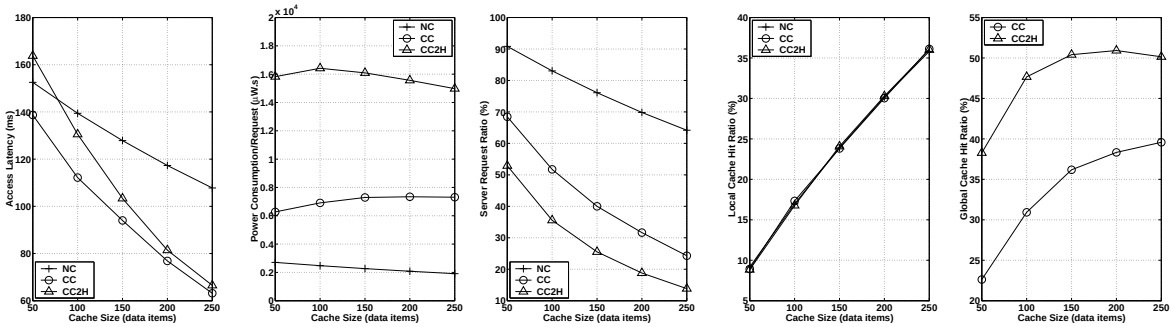
Continued on next page

Table 3.3 Simulation parameters and default settings. (Continued from previous page)

Parameter	Description	Default Value
$NumData$	No. of data items in the database	10,000 data items
$DataSize$	Data item size	4 kilobytes (KB)
$AccessInterval$	Mean think time between two consecutive requests	1 s
$AccessRange$	Client data access range	10% of $NumData$
$CommonAccessRange$	Common access range of all MHs	100% of $AccessRange$
$CacheSize$	Client cache size	100 data items
$Speed$ ($v_{min} \sim v_{max}$)	Mobility speed	$1 \sim 5$ m/s
T_{beacon}	Time interval between two consecutive beacons	1 s
τ_b	Timeout period for link failure detection	$2 \times T_{beacon}$
$TranRange$	Transmission range	100 m
$PauseTime$	Pause time used in "random-waypoint" mobility model	1 s
P_{disc}	The probability of a client disconnecting from the network	0
$DisconnectTime$	Mean disconnection time for an MH	10 s
ω	Weight used in Equation 3.1	0.25
φ, φ'	Network congestion parameters	10, 3
θ	Zipf skewness parameter	0.5

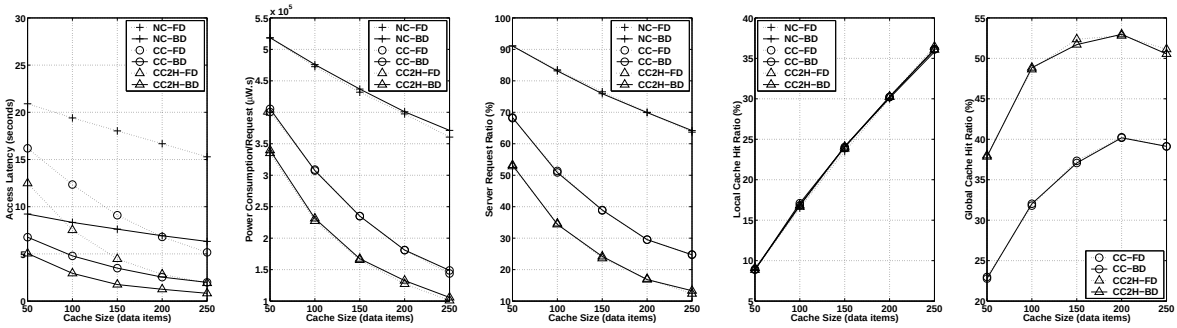
3.5.1 Effect of Cache Size

The first experiment studies the effect of cache size on system performance by varying the cache size from 50 to 250 data items.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

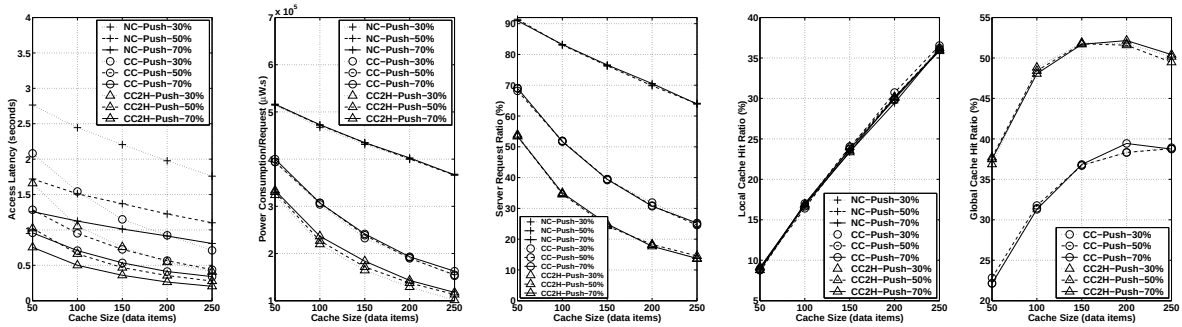
Figure 3.6: Effect of cache size in a pure pull-based environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.7: Effect of cache size in a pure push-based environment.

Figures 3.6(a), 3.6(c), 3.7(a), 3.7(c), 3.8(a) and 3.8(c) show that all schemes exhibit better access latency and server request ratio, as cache size gets larger. The MHs can cache more data items in the local cache with increasing cache size, so it improves the LCH ratio, as shown in Figure 3.6(d). For the MHs adopting COCA schemes, it not only achieves a higher LCH ratio as the cache size gets larger, but it also enjoys a higher GCH ratio. The GCH ratio of all schemes initially improves with increasing cache size, but the ratio drops when the cache size further gets larger. With larger cache space, the MHs can cache more data items, so that it increases the chance of finding some peers caching the required data items, i.e., the GCH ratio improves. However, when the cache size further increases, more requests can be satisfied by accessing to the cached data items. Therefore, the demand on the global



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.8: Effect of cache size in a hybrid environment.

cache decreases with increasing cache size, and hence the GCH ratio drops.

In the pull-based mobile environment, although CC-2H yields the highest GCH ratio, as shown in Figure 3.6(e), the access latency is worse than CC. In multi-hop data searching, the network traffic is a key factor to system performance. The MHs adopting CC-2H have to play a role as routers to forward requests and data items to other peers, so the multi-hop searching incurs a lot of network traffic. If the MHs cannot find the required data items within their neighboring peers, they better obtain the data items from the MSS.

However, in the push-based and hybrid mobile environments, CC-2H performs better than NC and CC in terms of access latency and server request ratio, as depicted in Figures 3.7(a) and 3.7(c) for the push-based environment, and Figures 3.8(a) and 3.8(c) for the hybrid environment. These two mobile environments are scalable, but the MHs generally suffer from longer access latency than in the pull-based mobile environment. As a result, in the push-based and hybrid mobile environments, the MHs can afford to request data items from their peers with a longer hop distance to improve access latency. In the hybrid environment, the performance of CC-2H and CC is consistently better than NC with different *PushChannel* ratio values.

The cost of adopting COCA schemes is higher power consumption, as shown in

Figures 3.6(b), 3.7(b) and 3.8(b). When the MHs enjoy a higher LCH ratio, they can reduce the power consumption searching in the Peer Cache layer. However, when the GCH ratio increases with increasing cache size, the MHs have to consume more power to return the required data items to the requesting peers. In addition, the MHs have to spend more power on discarding unintended messages as they are residing in the transmission range of the source MH, destination MH or both. The result also shows that the power consumption of CC-2H is much higher than NC and CC. In the pull-based mobile environment, the power consumption of the MHs adopting CC-2H is two times more than with CC.

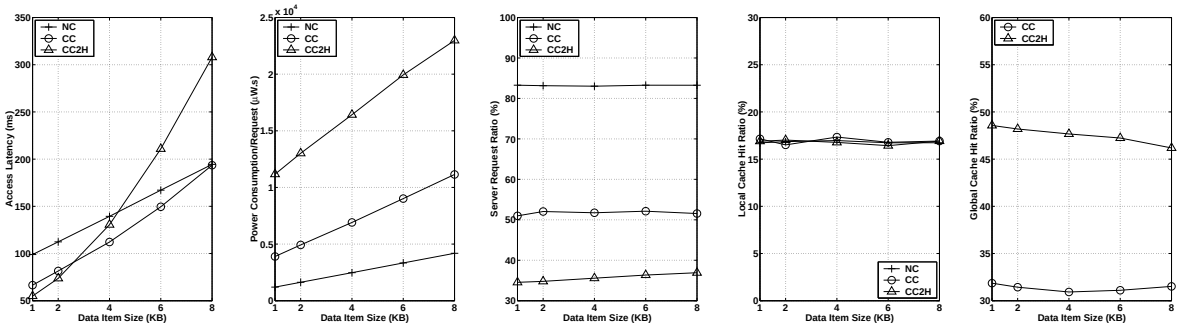
Figures 3.7(b) and 3.8(b) show that the MHs adopting CC-2H consume less power than the other schemes. When they encounter local cache misses, they tune in to the broadcast channel to catch the required data item or index. They can switch to doze mode until the data items are broadcast or obtain them via point-to-point channels if the data items are not allocated in the broadcast channel. However, the power that the MHs consume to wait for the index is significantly higher than searching among peers within a distance of two hops. Thus, it is worthy to study the system performance of multi-hop data searching with a distance of more than two hops; that will be investigated in Section 3.5.9.

3.5.2 Effect of Data Item Size

We then study the effect of data item size on system performance by varying the data item size from one to eight kilobytes (KB).

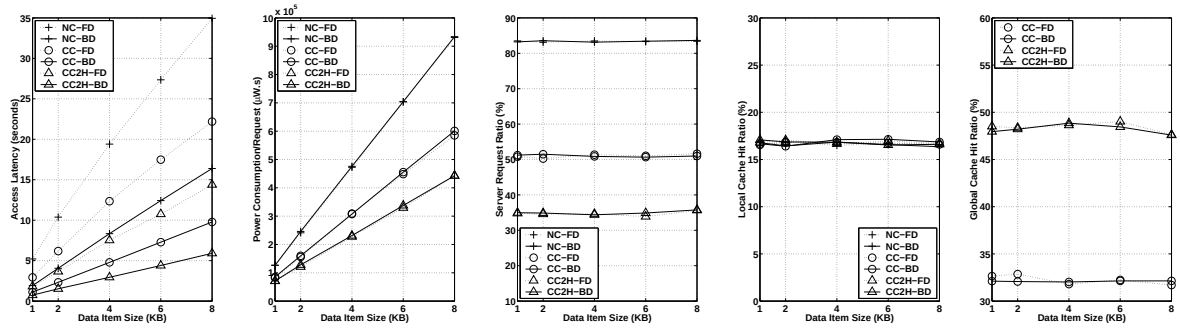
Figures 3.9, 3.10 and 3.11 show that access latency and power consumption increase with increasing data item size, since the transmission time of a data item increases as the data item size gets larger.

In the pull-based mobile environment, CC consistently performs better than NC



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.9: Effect of data item size in a pure pull-based environment.

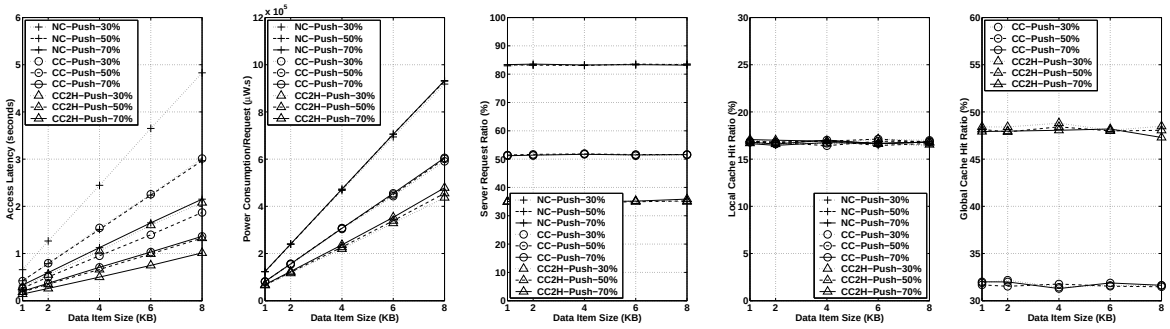


(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.10: Effect of data item size in a pure push-based environment.

in terms of access latency, as depicted in Figure 3.9(a). For CC-2H, it is the best when the data item size is small, i.e., two KB, but it performs worse than CC with increasing data item size. In CC-2H, the MHs have to forward the data items to the requesting peers, as the requesting peers cannot directly communicate with the target peers. When the data item size gets larger, the forwarding operation generates more network traffic and the requesting peers suffer from longer access latency. The longer hop distance between the requesting MHs and target peers also incur higher power consumption, as shown in Figure 3.9(b).

In the push-based and hybrid mobile environments, CC-2H gives the best performance in terms of access latency and power consumption, as shown in Figures 3.10(a) and 3.10(b) for the push-based environment, and Figures 3.11(a) and 3.11(b) for the

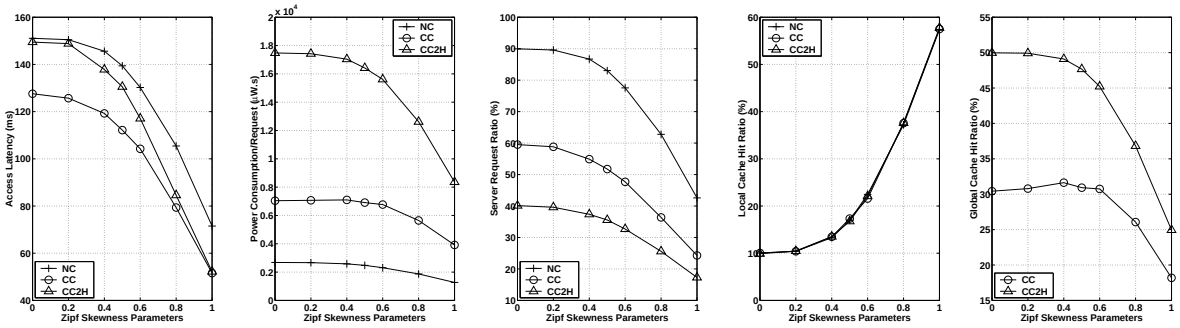


(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.11: Effect of data item size in a hybrid environment.

hybrid environment, because the latency and power consumption of access to data items from the MSS are much larger than that in the pull-based environment.

3.5.3 Effect of Skewness in Access Pattern

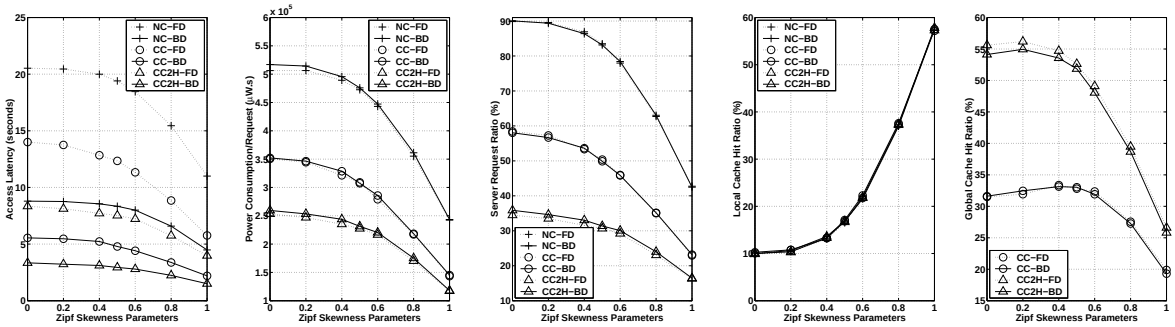


(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.12: Effect of skewness in access pattern in a pure pull-based environment.

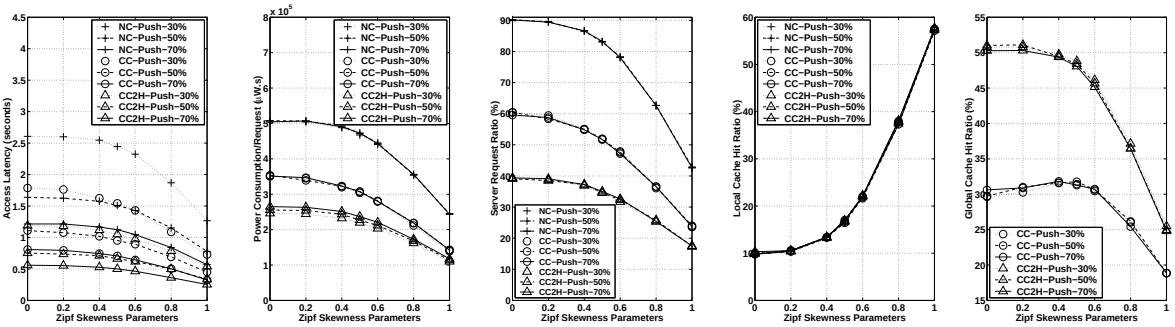
In this set of simulated experiments, we study the effect of skewness in access pattern by changing the skewness value from zero to one.

Figures 3.12, 3.13 and 3.14 show that system performance of all schemes improves with increasing skewness parameter value. When the skewness parameter is equal to zero, the MHs access the data items uniformly, i.e., they access all data items within the access range with the same probability. Thus, there is a higher chance for them to



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.13: Effect of skewness in access pattern in a pure push-based environment.



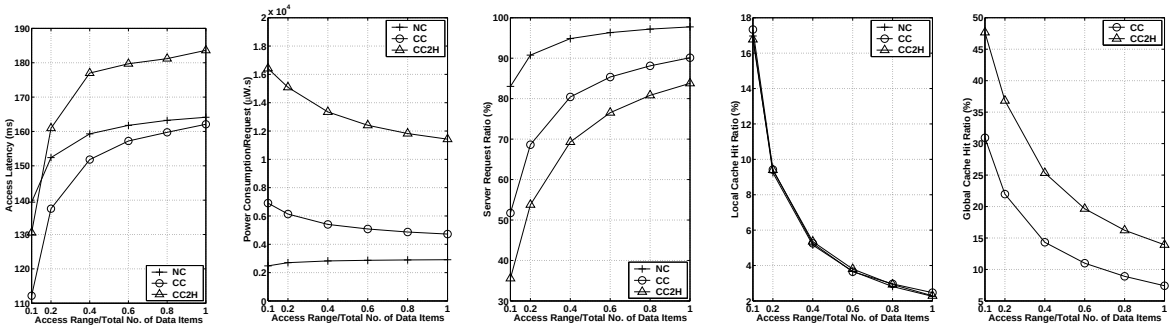
(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.14: Effect of skewness in access pattern in a hybrid environment.

encounter local cache misses, as shown in Figures 3.12(d), 3.13(d) and 3.14(d). When the skewness parameter increases, the client access pattern becomes more skewed, such that the MHs can find more required data items in their local cache. The higher the LCH ratio, the better the performance the MHs can achieve.

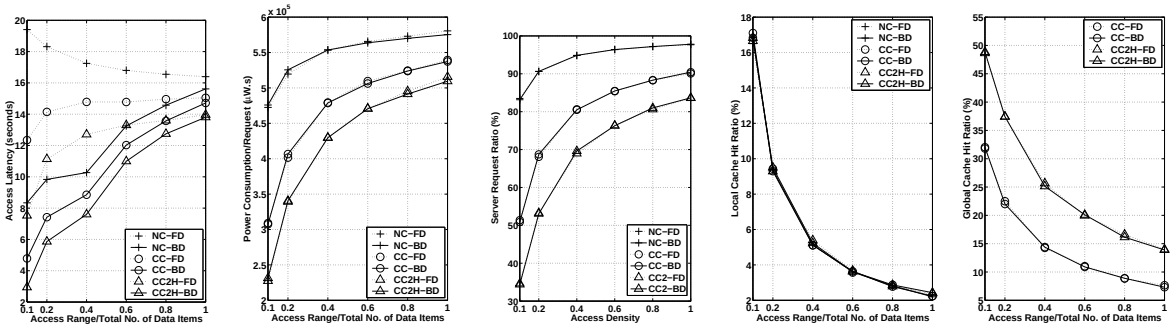
The power consumption of all schemes reduces as the skewness parameter gets larger. When the MHs achieve a higher LCH ratio, they can save on power consumption by not enlisting neither the MSS nor other peers for help. For COCA schemes, the MHs need less help from their peers, i.e., the GCH ratio reduces, with increasing LCH ratio.

3.5.4 Effect of Access Density



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.15: Effect of access density in a pure pull-based environment.

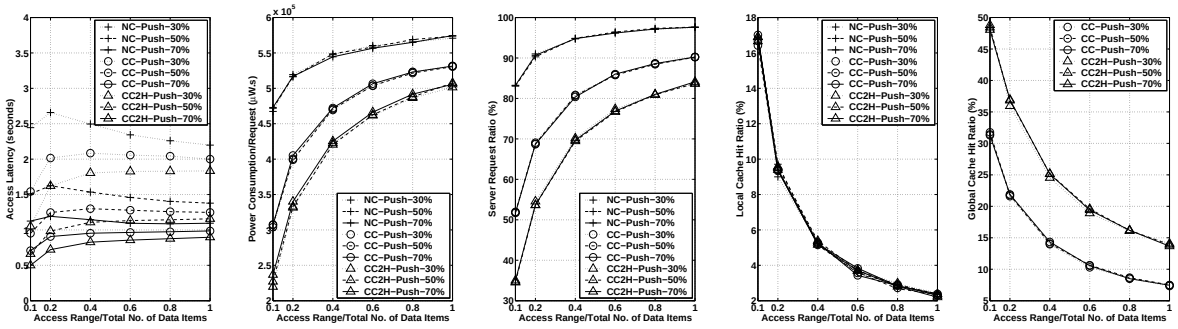


(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.16: Effect of access density in a pure push-based environment.

We study the effect of access density on system performance by changing the access range from 1000 to 10000 data items.

In the pull-based mobile environment, access latency and server request ratio increase with decreasing access density, i.e., increasing access range, as shown in Figures 3.15(a) and 3.15(c). The MHs access more distinct data items as the access density gets lower; they experience a higher chance to encounter local cache misses, as depicted in Figure 3.15(d). Likewise, when the MHs are interested in more distinct data items, the GCH ratio also reduces, as shown in Figure 3.15(e). Hence, more client requests are sent to the MSS, i.e., the server request ratio increases. As



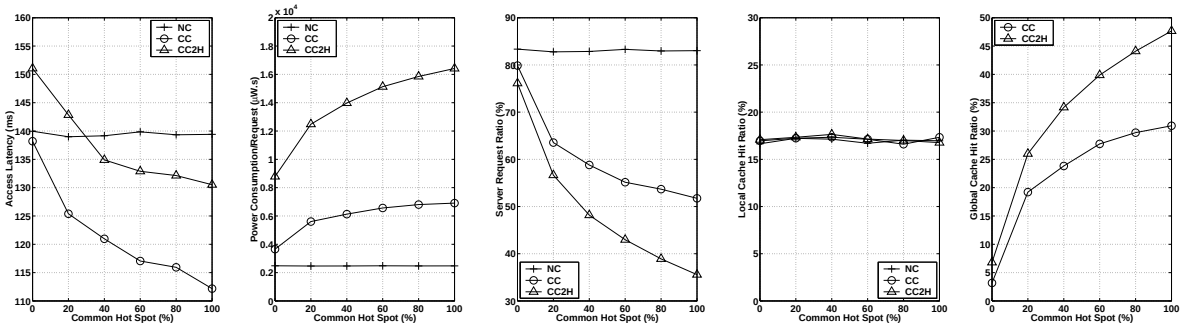
(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.17: Effect of access density in a hybrid environment.

fewer MHs can turn in the required data items to the requesting peers, the power consumption reduces, as exhibited in Figure 3.15(b). However, the MHs adopting NC consume more power, as they need to obtain more data items from the MSS. Although the MHs adopting COCA schemes also issue more requests to the MSS, due to the fact that the power consumption on obtaining data items from peers is much higher than doing so from MSS. Thus, the overall power consumption for CC and CC-2H reduces with increasing access range.

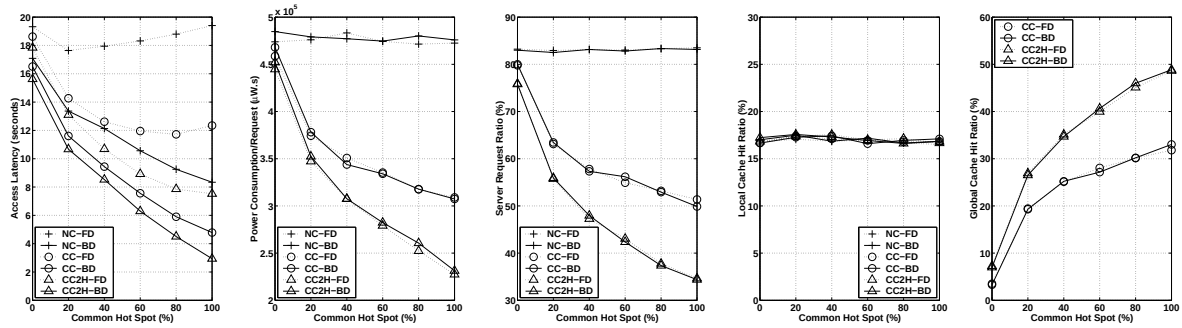
Figure 3.16(a) shows that the access latency of NC-FD decreases, as the access density gets lower. When the client access range contains more data items allocated in the broadcast channel, the access latency approaches the expected access latency. As the access range is equal to the total number of data items allocated to the flat broadcast disk, the expected access latency is at most half of the broadcast cycle, i.e., $\frac{NumData}{2} \times \frac{DataSize}{BW_{Server}} = \frac{10000}{2} \times \frac{32768}{10000000} = 16.384$ seconds, as reflected in Figure 3.16(a). In CC-BD and CC-2H-BD, when the client access range contains more data items that are allocated in the disk with the slowest spinning speed, i.e., the disk three, the access latency sharply increases. The gap between the performance of FD and BD closes with increasing access density.

3.5.5 Effect of Common Hot Spot



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.18: Effect of common hot spot in a pure pull-based environment.

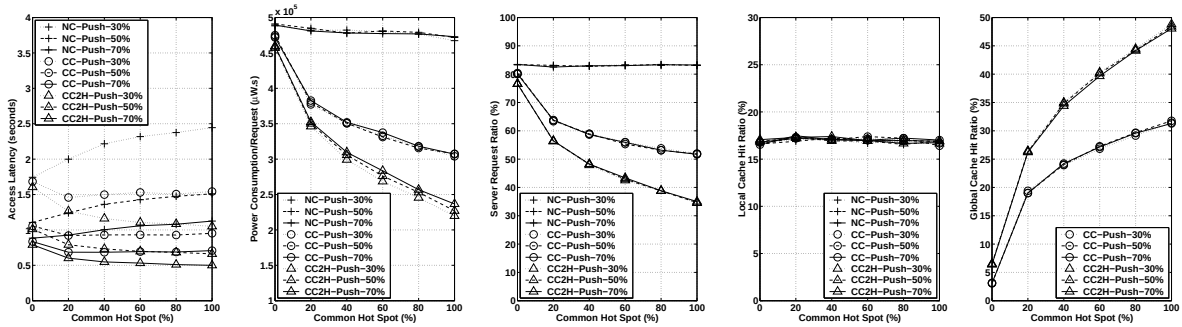


(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.19: Effect of common hot spot in a pure push-based environment.

The effect of the percentage of common hot spot of all MHs on system performance is studied by varying the percentage from 0 to 100 percent.

In the pull-based mobile environment, the varying percentage of common hot spot does not affect NC scheme. However, it is obvious that the access latency and server request ratio of COCA schemes improve with increasing percentage of common hot spot. When the MHs share more common access pattern, the chance of getting the required data items from their peers increases. As shown in Figure 3.18(e), the GCH ratio improves with increasing percentage of common hot spot among MHs. The cost of obtaining data items from peers is higher power consumption. The power



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.20: Effect of common hot spot in a hybrid environment.

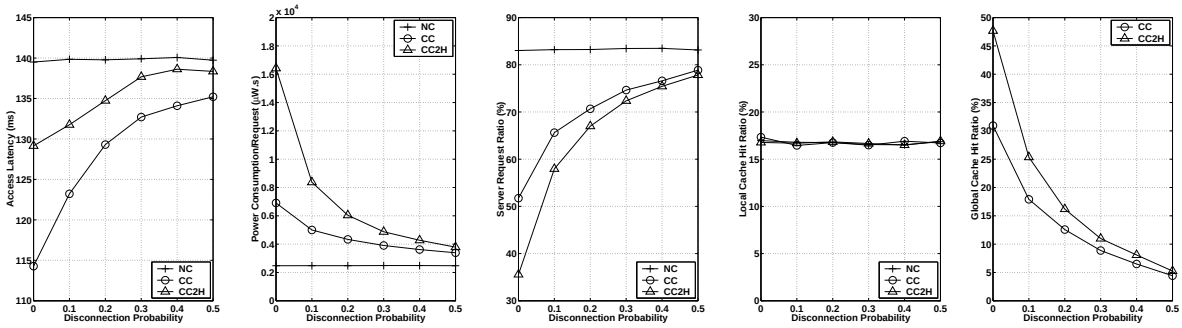
consumption of CC and CC-2H gets higher, as the GCH ratio increases, as depicted in Figure 3.18(b). Since the MHs adopting CC-2H can search outside of its transmission range, so they enjoy a higher GCH ratio than those with CC, but they have to spend more power on searching the Peer Cache layer as well.

The worst case of COCA schemes occurs when the client access range is completely random. Since the MHs do not have any knowledge of the access pattern of other peers, they always search the Peer Cache layer when they encounter local cache miss. The MHs have to wait for a timeout to detect a global cache miss, and then they enlist the MSS for help. Therefore, if there is a high global cache miss ratio, the MHs suffer from a longer access latency. When the GCH is low, the performance of CC-2H is worse than NC, as shown in Figure 3.18(a).

In the push-based and hybrid mobile environments, CC-2H exhibits the best access latency, server request ratio and power consumption compared with NC and CC. It is due to the fact that the access latency and power consumption of a global cache access is much less than accessing data items from the broadcast channel. Thus, it is worthy to search the required data items in further peers' cache to achieve better access latency and power consumption in these mobile environments. We will investigate the effect of hop distance on system performance in Section 3.5.9.

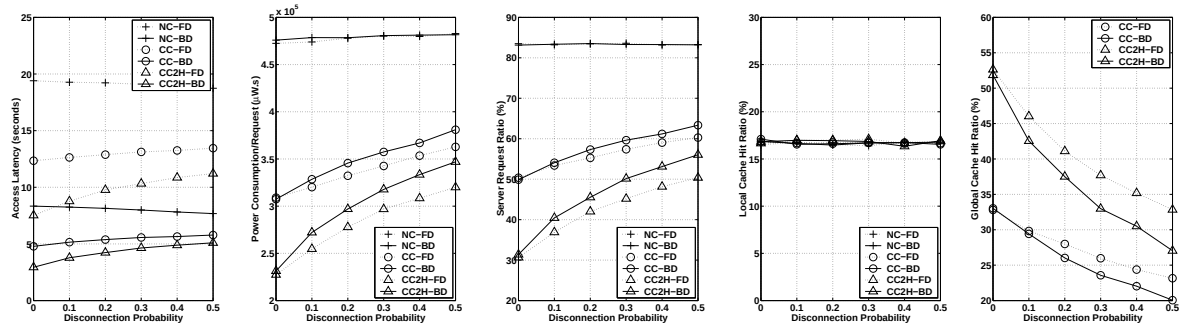
Figure 3.20(a) shows that the access latency of NC-Push-30%, NC-Push-50% and NC-Push-70% gets larger with increasing common hot spot. When the common hot spot increases, the range of the hot spot becomes narrower. As a result, more hot data items can be allocated to the broadcast channel. However, the latency of accessing data items in the broadcast channel is longer than that in a the pull-based manner, so the MHs suffer from a longer access latency when they often wait for their desired data items to appear in the broadcast channel.

3.5.6 Effect of Client Disconnection Probability



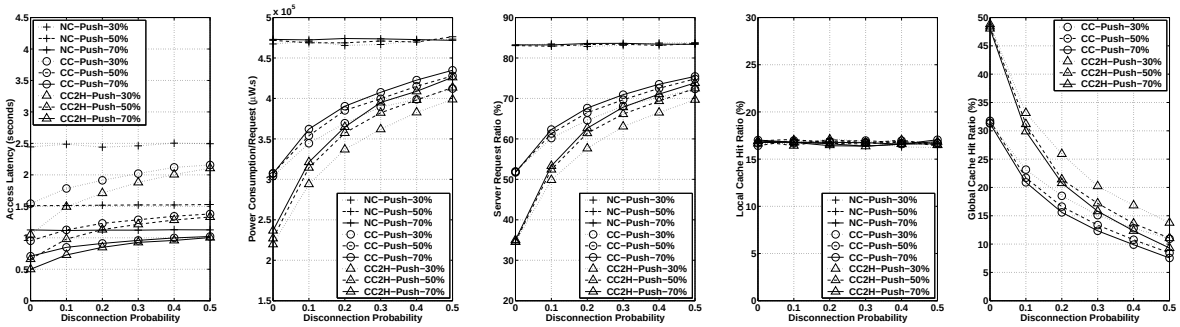
(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.21: Effect of client disconnection probability in a pure pull-based environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.22: Effect of client disconnection probability in a pure push-based environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.23: Effect of client disconnection probability in a hybrid environment.

In this set of simulated experiments, we study the effect of client disconnection probability on system performance by varying the probability from 0 to 0.5.

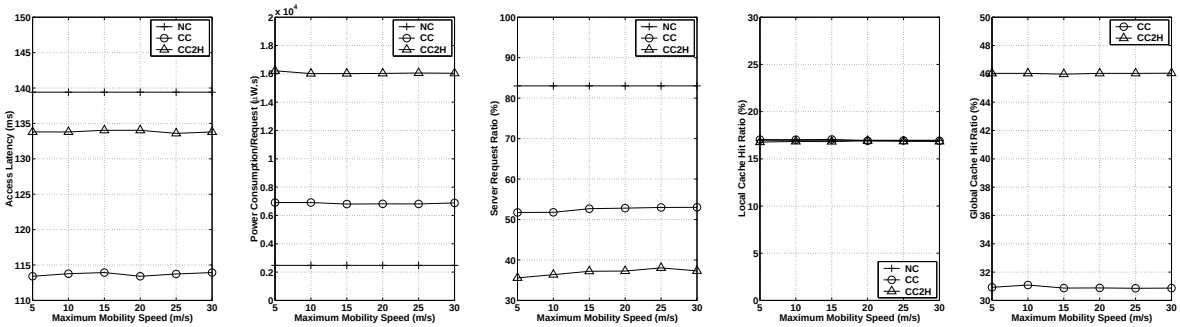
The result shows that the performance of COCA schemes degrades with increasing client disconnection probability. The chance of obtaining the required data items from peers decreases, as there are more disconnected peers in the system. As a result, the access latency and server request ratio increases for the MHs adopting CC and CC-2H, as the client disconnection probability gets higher.

In the pull-based mobile environment, the power consumption of the COCA scheme decreases with increasing client disconnection probability, as shown in Figure 3.21(b). When the GCH ratio decreases, more data items are retrieved from the MSS, so the MHs can save on more power, due to the fact that power consumption on retrieving data items from the MSS is much lower than doing so from other peers.

In the push-based and hybrid mobile environments, the power consumption of the COCA schemes is found to rise with increasing client disconnection probability, as shown in Figures 3.22(b) and 3.23(b). As the access latency and power consumption of a global cache access is much lower than accessing data items from the broadcast channel, when the MHs adopting CC and CC-2H encounter a higher global cache miss ratio with increasing client disconnection probability, they have to spend more power

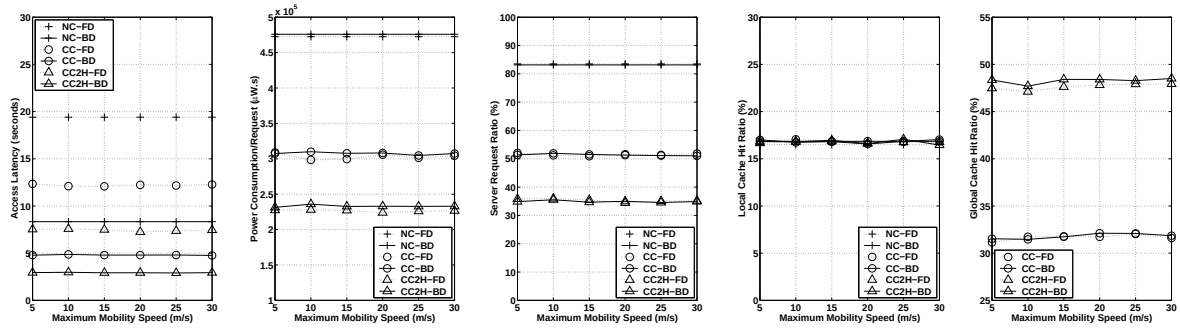
on obtaining their desired data items from the broadcast channel. In CC-2H, the MHs enjoy a higher GCH ratio, as shown in Figures 3.22(e) and 3.23(e); they can thus save more power than those in CC.

3.5.7 Effect of Mobility Speed



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.24: Effect of mobility speed in a pure pull-based environment.

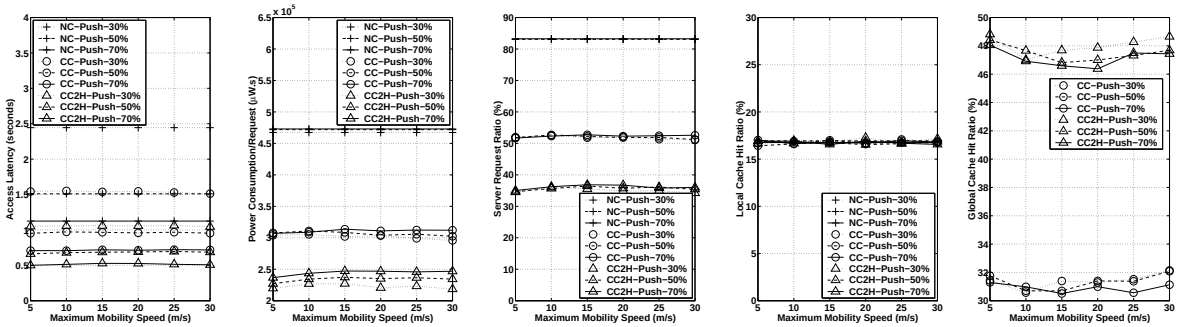


(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.25: Effect of mobility speed in a pure push-based environment.

In this experiment, we study the effect of mobility speed on system performance by increasing the maximum mobility speed from 5 m/s to 30 m/s.

As depicted in Figures 3.24, 3.25 and 3.26, the performance of NC is not affected by varying the mobility speed, and the performance of CC and CC-2H schemes is merely slightly affected by varying the mobility speed. In the pull-based environment, CC



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.26: Effect of mobility speed in a hybrid environment.

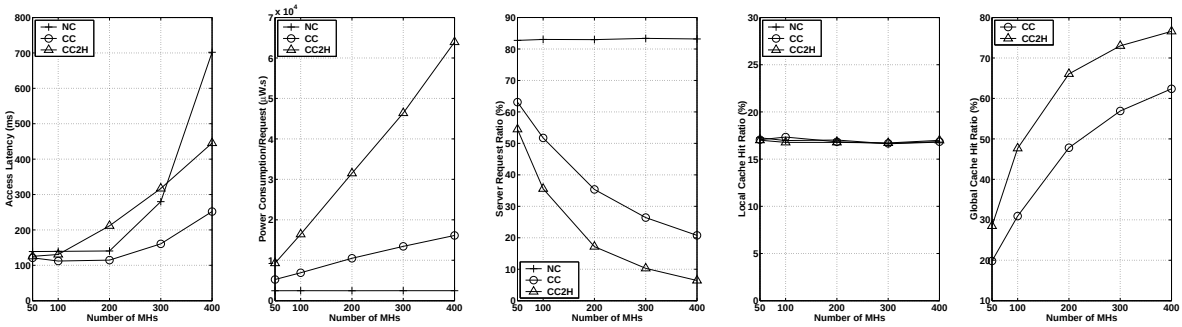
performs better than NC in terms of access latency and power consumption. However, CC-2H exhibits the best performance in server request ratio, as it achieves a higher GCH ratio than CC, as shown in Figure 3.24(e).

In the push-based and hybrid environments, GCH ratio is a crucial factor to system performance, as it effectively reduces access latency and power consumption. CC-2H consistently performs better than the other two schemes because CC-2H gives higher GCH ratio in the push-based and hybrid environments, as depicted in Figures 3.25(e) and 3.26(e). Based on the result of this series of simulated experiments, CC and CC-2H schemes are well adapted to the various speed levels in the pull-based, push-based and hybrid environments.

3.5.8 Effect of Number of MHs

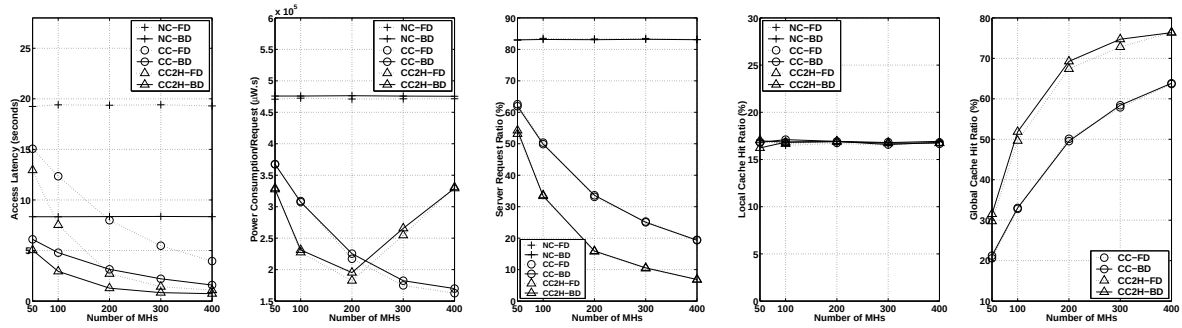
This experiment studies the effect of client population on system performance by increasing the number of MHs in the system from 50 to 400.

In the pull-based mobile environment, the access latency of all schemes increases with increasing number of MHs in the system, as shown in Figure 3.27(a). In terms of access latency, CC outperforms NC and CC-2H, and CC-2H performs better than NC in heavily-loaded environments. The COCA schemes can improve access latency



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.27: Effect of number of MHs in a pure pull-based environment.

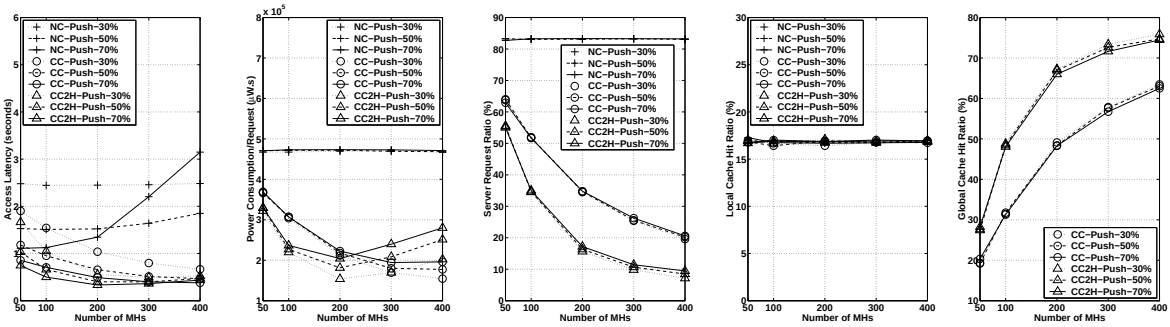


(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.28: Effect of number of MHs in a pure push-based environment.

because a large number of requests are handled by the MHs, and thus the system workload is effectively distributed among MHs. When an MH can reach more peers with increasing client population in the system, it stands a higher chance to obtain the required data items, and therefore, the GCH ratio increases as the number of MHs gets larger, as shown in Figure 3.27(e). However, since the MHs have to spend more power to handle global cache queries, such as receiving more broadcast queries from peers and forwarding more data items to requesting MHs, the power consumption increases with increasing GCH ratio, as illustrated in Figure 3.27(b).

In the push-based environment, the access latency of NC-FD and NC-BD is not affected by the number of MHs in the system. However, the access latency of the COCA schemes improves significantly with increasing number of MHs, as depicted in



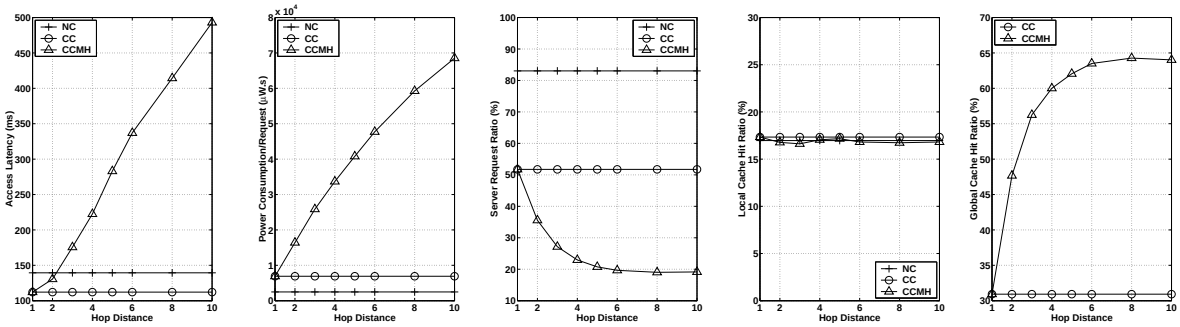
(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.29: Effect of number of MHs in a hybrid environment.

Figure 3.28(a), because the MHs adopting the COCA schemes achieve a higher GCH ratio with increasing client population, and coupled with the fact that the latency and power consumption of a global cache access is much lower than a broadcast channel access. The power consumption of CC-2H initially drops and then rises with increasing client population in the system. The initial drop is due to the fact that there is a higher chance for the MHs to obtain their desired data items from the peers as the power consumption of a global cache access is much less than downloading the required data items from the broadcast channel, so that they can conserve more power. However, when the number of MHs further increases, the power consumption on P2P communication offsets the benefit of achieving a higher GCH ratio, as the MHs have to receive more broadcast requests from peers, to return more data items to the requesting MHs and to discard a larger amount of unintended messages.

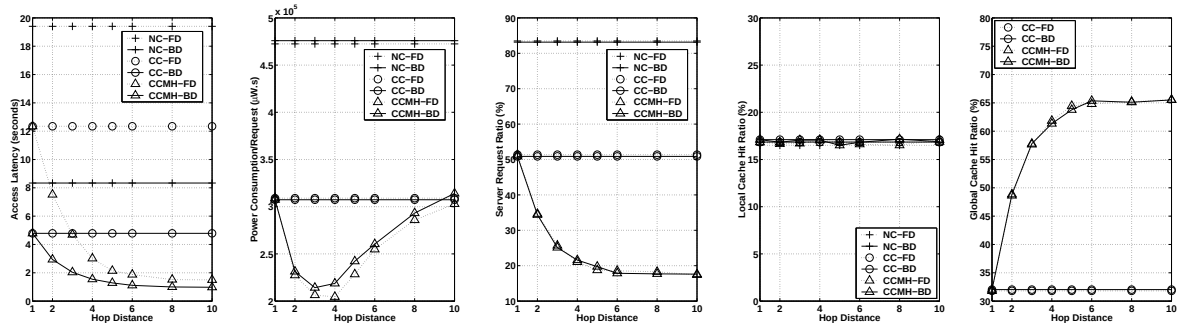
3.5.9 Effect of Hop Distance

Finally, we study the effect of hop distance on system performance by adjusting the hop distance from 1 to 10. Note that the MHs adopting CC only search for the desired data items in the cache of their neighboring peers, i.e., the hop distance is constantly equal to one, and no client cooperative is involved in NC.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.30: Effect of hop distance in a pure pull-based environment.

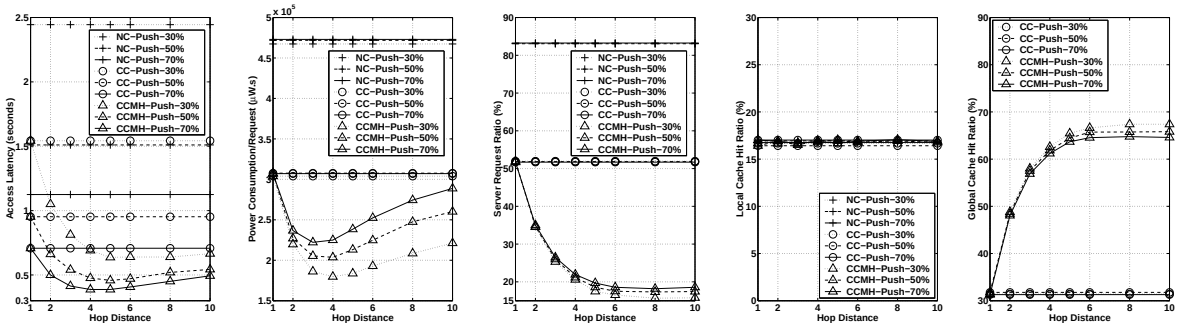


(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.31: Effect of hop distance in a pure push-based environment.

The effect of hop distance in the pull-based mobile environment is shown in Figure 3.30. Although the GCH ratio improves with increasing hop distance, the access latency and power consumption also increase, as the MHs have to spend more time and consume more power on searching for the desired data items in the Peer Cache layer. Combined with the result from Figure 3.27, the MHs should only apply multi-hop searching in the pure-pull mobile environment when the system is highly overloaded, and the hop distance should not exceed two, in order to conserve battery power.

Figure 3.31 exhibits the result of varying hop distance in the mobile broadcast environment. The access latency of CC-MH-FD and CC-MH-BD improves with increasing hop distance in searching for desired data items in the system. When the hop distance gets larger, there is a higher chance for the MHs to obtain the desired



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 3.32: Effect of hop distance in a hybrid environment.

data items from their peers, as shown in Figure 3.31(e). We observe that retrieving data items from peers is more efficient than passively waiting for them in the broadcast channel in terms of access latency. The power consumption of CC-MH-FD and CC-MH-BD first reduces and then rises with increasing hop distance, as depicted in Figure 3.31(b). The initial drop is due to the fact that the MHs can save on power when they access the desired data items from their peers. However, if the MHs search for the data items in a distance of larger than three hops, the power consumption increases as the hop distance gets larger. This is because the power consumption on disseminating request messages to distant peers and forwarding requested data items from the distant peers to the requesting MHs undermines the benefit of multi-hop searching. Figure 3.31(e) shows that the GCH ratio is saturated as the hop distance is over six. Therefore, we suggest that the MHs should merely search for their desired data items within six hops, in order to gain benefit of multi-hop searching in terms of access latency and power consumption.

The effect of hop distance in the hybrid environments is similar to the push-based environments, as depicted in Figure 3.32. Figure 3.32(e) shows that the GCH ratio is saturated at a distance of six hops. We suggest that the MHs should not search for their desired data items from the cache of the distant peers, i.e., the hop distance

is larger than six hops, in order to optimize system performance in terms of access latency and power consumption, as the multi-hop searching traffic can increase access latency when the hop distance is getting further, as shown in Figure 3.32(a).

3.6 Concluding Remarks

In this chapter, we have proposed a mobile cooperative caching for MHs, called COCA, in mobile environments. Three COCA models are described for three different mobile data dissemination models: pull-based, push-based and hybrid. To extensively evaluate the performance of COCA, a multi-hop searching protocol is also described. The performance of COCA is evaluated through a series of simulated experiments by considering various client behaviors and system properties. The results show that COCA with single-hop searching always performs better than traditional caching scheme in the pull-based environment in terms of access latency and server request ratio. However, the cost of MHs adopting COCA is higher power consumption. In the broadcast and hybrid mobile environments, COCA yields the better performance in access latency, power consumption and server request ratio compared with the conventional caching scheme, as the MHs can achieve a higher GCH ratio. COCA with multi-hop searching further improve system performance in the push-based and hybrid environments.

Based on the simulated experiment results, we conclude that the MHs should not search too far away in the pull-based environment, as cost of multi-hop searching is more expensive than getting the required data items from the MSS. However, they better enlist more peers with the distance of a maximum number of six hops for help in the push-based and hybrid environments because the cost of retrieving data items from MSS via broadcast channels is higher than obtaining them from peers with a

distance of multi-hop.

The results also show that COCA can improve the system scalability, as the system workload can be shared among MHs. COCA is good for the pull-based mobile environment; on the other hand, COCA with multi-hop searching is more suitable for push-based and hybrid mobile environments.

Chapter 4

Cache Signature Scheme

4.1 Introduction

As the MHs do not possess any knowledge about the cache content of their peers, they have to search their peers' cache whenever they encounter local cache misses. In case of no peer caching the required data items, the search in the Peer Cache layer gives in vain, wasting power and increasing access latency. In this chapter, a cache signature scheme is proposed for the MHs. It provides hints for the MHs to make decision whether to search the Peer Cache layer based on their local state. In the cache signature scheme, each signature is stored as a bloom filter [15], which will be described in the next section.

Signature techniques have been applied in information filtering in mobile broadcast environments [59]. In [59], signatures are used to summarize the information that will be broadcast in next information segment. When an MH encounters a local cache miss, it generates a query signature, and then tunes in to the broadcast channel to download the signature from the MSS. If the signature matches the query signature, the MH keeps listening to the broadcast channel and catches the required data items.

Otherwise, it switches to the doze mode to reduce power consumption until the MSS broadcasts the next signature.

Signature techniques are also applied on efficiently searching P2P networks [62]. In [62], the clients exchange their local signatures with other peers within their neighborhood, and then the clients are able to determine whether they should search outside of their neighborhood, or select a subset of neighborhood and forward the request to them, in order to reduce network traffic.

In [34], signatures are adopted to facilitate URL (Uniform Resource Locator) routing in WWW (World Wide Web) content distribution networks. The author in [34] proposes to use signature as a compression method to reduce variable length URLs to fixed length integer signatures. The signatures that are generated by cyclic redundancy check (CRC) codes are used as keys in the URL routing table. The trace-driven simulation result shows that using signatures can reduce the routing table size and improve the look-up time.

Signature techniques are also used in web cache sharing protocol [30]. In [30], each proxy maintains a signature that summarizes its cached data items. The proxies can determine by exchanging the signatures among themselves which proxies are likely to cache the required data items, and then forward the requests to them. It also proposes a proactive approach to generate signatures to reduce searching latency and processing computation overheads.

A compressed bloom filter scheme [65] is proposed, that compresses the bloom filter with arithmetic coding to reduce its size before transmitting to others, so as to improve system performance.

In our work, we adopt signature techniques in a mobile cooperative caching environment. A proactive signature generation mechanism is used to reduce processing computation overhead and response time. To reduce the transmission time and power

consumption, signatures are compressed before sending to other peers. Three signature storage schemes are proposed to study the tradeoff between storage space and maintenance overhead. A cache signature exchange protocol is also proposed for the MHs to exchange their cache signature with their neighbors. The exchange protocol is then extended to handle client disconnection by adopting a time-stamping technique.

4.2 Background

A bloom filter is used to represent a set $S = \{s_1, s_2, \ldots, s_N\}$ of N elements (also called keys). A vector V with M bits, denoted $v_1, v_2, \ldots, v_M$, is initially set to zero. In addition, there are k independent hash functions, denoted $h_1, h_2, \ldots, h_k$, each yielding a hash value within the range between 0 and $M - 1$.

To construct a bloom filter, an element $s \in S$ is hashed by the k hash functions to produce k hash values, $h_1(s), h_2(s), \ldots, h_k(s)$. Then, the corresponding bits in V for the hash values are set to one, i.e., $\forall j \in [1, k], v_{h_j(s)} = 1$. This two-step procedure is repeated for each element in S. To check whether an element s' is contained in V, s' is first hashed by the k hash functions. If all the bits at positions, $h_1(s'), h_2(s'), \ldots, h_k(s')$, are set in V, i.e., $\forall j \in [1, k], v_{h_j(s')} = 1$, then s' is probably stored in V. Otherwise, if no bit or only some of the bits are set in V, i.e., $\exists j \in [1, k], v_{h_j(s')} = 0$, it indicates that s' must not be stored in V.

Since each bit in V can be randomly set multiple times by different elements, a bloom filter, i.e., the vector V, could give a wrong indication, called a *false positive*, that occurs when the bloom filter indicates the existence of an element, but the element is actually not stored in it.

The false positive probability is dependent on the values of k, M and N. After hashing all elements in S into V, the probability that a particular bit still remains

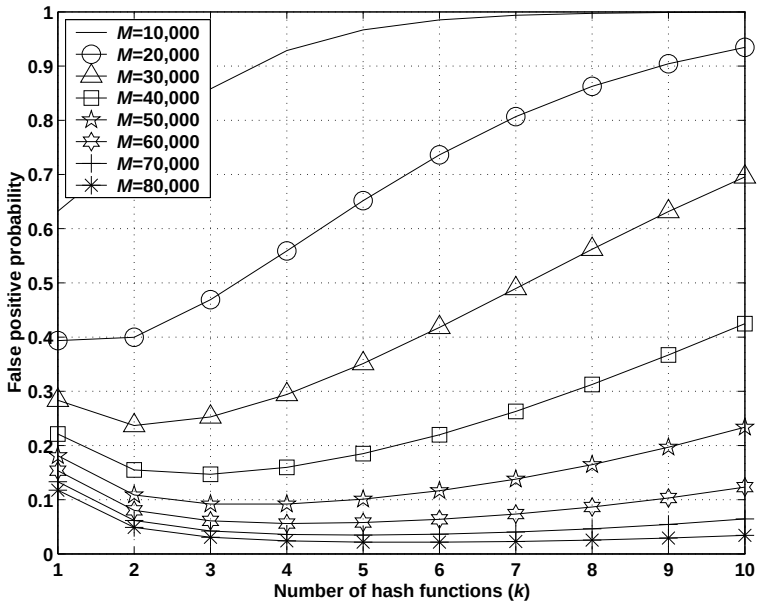


Figure 4.1: The false positive probability with different values of k and M. ($N = 10,000$)

zero is $(1 - 1/M)^{Nk}$. It is commonly assumed that all bits are independently set to zero and one [30, 65]. Thus, the probability of the occurrence of a false positive, i.e., an element is not stored in V, but all the corresponding bits are set, can be calculated by the equation:

$$\left(1 - \left(1 - \frac{1}{M}\right)^{Nk}\right)^{k}, \tag{4.1}$$

where k, M and N are positive integers. Figure 4.1 exhibits the false positive probability with different values of M and k if $N = 10000$. An approximate optimal number of hash functions, k, to produce the lowest false positive probability with respect to the values of M and N is derived in [30], i.e., $k = (\ln 2)(M/N)$.

4.3 Cache Signatures

In the cache signature scheme, there are four types of signatures, *data signature*, *cache signature*, *peer signature* and *search signature* [22]. A data signature for a data item is a bit string that is produced by hashing its unique identifier, such as integral identifier, keywords, attribute values, URL, document content with single or multiple hash functions. A cache signature is a bloom filter that summarizes the cache content by superimposing all data signatures of the data items in a cache. A peer signature for an MH is produced by superimposing the cache signatures of its peers to provide hints for the MH to determine whether to search the Peer Cache layer, when it encounters a local cache miss. The generation procedure of a data signature and search signature is the same. A data signature is for a cached data item, while search signature is for a requested data item that cannot be found in the local cache. The search signature is used for comparing with the peer signature by a bitwise AND operation. If the result is the same as the search signature, it indicates that some of peers are likely caching the required data item, so the MH searches the Peer Cache layer. Otherwise, the MH will bypass the Peer Cache layer and obtain the required data item from the MSS directly instead.

All MHs need to exchange their own cache signature with other peers. To enable them to manipulate the cache signature of their peers, the bloom filter size should be identical across all participating peers, even though they may only cache a portion of data items in the system. Furthermore, in a highly dynamic distributed environment, it is impossible to limit or predict the client cache size, so we have to assume that some MHs could possess very large storage capacity that is enough to cache most, if not all, data items of the system. To this end, when we determine the value of k, the total of number of data items of the system rather than the client cache size will be considered as the setting of M.

4.3.1 Cache Signatures with Compression

When an MH merely caches a small portion of data items of the system, there are many "zeros" in its cache signature; therefore the transmission overhead can be reduced, if the bloom filter is compressed before it is transmitted to other peers. A variable-length-to-fixed-length (VLFL) run-length encoding [10] is used to compress the cache signature because it is simple and yet practical for most mobile devices. The VLFL run-length encoding consists of two steps. First, the sequence of bits is decomposed into run-lengths (L) which are terminated by two cases:

1. there are R consecutive "zeros", where $R = 2^l - 1$ for any positive integer l;
2. there are L consecutive "zeros" followed by a single "one", where $0 \leq L < R$.

Second, a fixed-length codeword, which indicates the length of the run-length, is assigned to each run-length.

The compression performance is dependent on the value of R, as it defines the maximum run-length in the first step of the VLFL run-length encoding. An analytical model has been developed to select an optimal value for R [10]. The probability of the occurrence of a "zero" in a signature is $p_0 = (1 - 1/M)^{Nk}$. In addition, the probabilities of possible run-lengths, $0 \leq L \leq R$ can be obtained by:

$$P(L) = \begin{cases} p_0^L(1 - p_0), & \text{for } 0 \leq L < R \\ p_0^R, & \text{for } L = R \end{cases} \tag{4.2}$$

Hence, the expected length of an intermediate symbol, that is either a run-length with R consecutive "zeros" or a run-length with L consecutive "zeros" and a terminator "one", where $0 \leq L < R$, is:

$$\eta = \sum_{L=0}^{R-1}(L+1)P(L) + RP(R) = \frac{1 - p^R}{1 - p}. \tag{4.3}$$

Let M' be the length of a compressed signature. The expected length of a compressed signature, $E(M')$, is given by:

$$E(M') = (M/\eta) \times \log_2(R+1), \tag{4.4}$$

where M is the length of a cache signature. Algorithm 3 is used to determine an optimal value of R based on the values of N, M and k. We can take advantage of the VLFL encoding compression, i.e., $M' < M$, when $\log_2(R+1) < \eta$. Therefore, an MH has to make a local decision on whether to compress its cache signature before transmitting it to other peers based on three factors: its cache size ($|Cache|$) in terms of number of data items that can be stored in the cache[1], bloom filter size (M) and the number of independent hash functions (k). The VLFL encoding compression only reduces a cache signature size, as the expected length of an intermediate symbol is shorter than the length of a fixed-length codeword, i.e., $\log_2(R+1) < \eta$. A cache signature thus should be compressed, if the condition, $\log_2(\widehat{R}+1) < \frac{1-\rho^{\hat{R}}}{1-\rho}$, where $\widehat{R} = \mathsf{FindOptimalR}(|Cache|, M, k)$, is satisfied. Otherwise, the MH simply sends its cache signature to other peers without any compression.

Table 4.1 shows the expected length of compressed cache signatures with different values of M and R for MHs with the cache size of N data items, and the highlighted figures are the compressed signature with the shortest length for different values of M, and R in the column of a highlighted figure is the optimal value for the corresponding value of M in the row.

[1]If the size, *data_size*, of all data items is identical, the number of data items can be stored in a cache is equal to $\frac{cache\_size}{data\_size}$, where *cache_size* is the cache size of an MH; otherwise, it is equal to the average number of data items that can be cached, i.e., $\frac{cache\_size}{\frac{1}{N}\sum_{i=1}^{N} data\_size_i}$.

Algorithm 3 Finding an optimal value of R.

1: **FindOptimalR** (N, M, k)
2: $\rho \leftarrow \left(1 - \left(\frac{1}{M}\right)^{Nk}\right)$;
3: $min\_M' \leftarrow +\infty$;
4: $R \leftarrow 1$;
5: $i \leftarrow 1$;
6: $Optimal\_R \leftarrow 1$;
7: **while** $i \leq \frac{1-\rho^R}{1-\rho}$ **do**
8: $tmp\_M' \leftarrow \frac{Mi(1-\rho)}{1-\rho^R}$;
9: **if** $tmp\_M' < min\_M'$ **then**
10: $min\_M' \leftarrow tmp\_M'$;
11: $Optimal\_R \leftarrow R$;
12: **else**
13: break;
14: **end if**
15: $i \leftarrow i + 1$;
16: $R \leftarrow 2^i - 1$;
17: **end while**
18: return $Optimal\_R$;

4.3.2 Generation of Cache Signatures

The generation mechanism of cache signature is similar to that of a summary cache [30]. The identifier of a data item is first hashed by a 128-bit MD5 [70] signature. The 128-bit hash value is then divided into four 32-bit words. A modulus function is next applied on each 32-bit word to yield a hash value bounded by M (the vector size), i.e., between zero and $M - 1$, as illustrated in Figure 4.2. If $k = 4$, the bits at the position of the four hash values are set to one in the vector. This process is repeated for each cached data item. Then, the cache signature is compressed by the VLFL run-length encoding. If $k < 4$, only the bits at the position of the first k hash values are set to one; or if $k = 1$, cyclic redundancy check (e.g., CRC32) [34] can be used to generate a single 32-bit word. In case of $k > 4$, other additional independent hash functions would be used, such as CRC32, some secure hash algorithms (e.g., SHAs) and so on.

Table 4.1: The expected length of compressed cache signatures with different values of R and M ($N = 100$, $k = 2$).

M	R									
	1	**3**	**7**	**15**	**31**	**63**	**127**	**255**	**511**	**1023**
10000	10000	6800.4	4547.1	3056	2142.8	1658.6	**1504.8**	1593.9	1782.3	1980.2
20000	20000	13466.9	8830.7	5714.7	3732.9	2554.6	1937	**1726.9**	1801.9	1990.2
30000	30000	20133.5	13115.7	8378.7	5338.1	3487.4	2443	1951.1	**1855.6**	1995.6
40000	40000	26800.1	17401.1	11044	6947.1	4429.9	2970.9	2214.9	**1946.8**	2007.1
50000	50000	33466.8	21686.6	13709.9	8557.7	5376.3	3507.9	2497.3	2063.7	**2029.9**
60000	60000	40133.4	25972.1	16376	10169.1	6324.7	4049.6	2789.7	2197	**2064.9**
70000	70000	46800.1	30257.8	19042.3	11780.9	7274.3	4593.9	3088	2341.1	**2110.7**
80000	80000	53466.7	34543.4	21708.7	13392.9	8224.5	5140	3390	2492.5	**2165.3**
90000	90000	60133.4	38829	24375.1	15005.2	9175.2	5687.2	3694.6	2649	**2227.1**
100000	100000	66800	43114.7	27041.6	16617.6	10126.3	6235.2	4001	2809.1	**2294.6**

4.4 Cache Signatures with Proactive Generation

Since the cache content is changed frequently, i.e., every cache insertion and eviction changes the cache signature, the MH has to re-generate the cache signature before

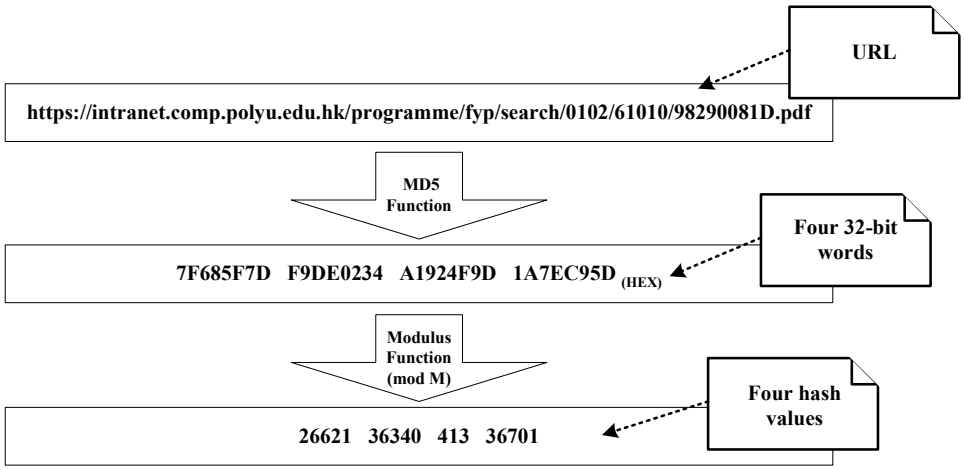


Figure 4.2: An example of the generation of a data signature from a URL.

sending it to other peers. To reduce the processing computation overhead, a proactive approach is used to generate cache signatures. In the proactive approach, a counter vector [30] is adopted to maintain the information of a cache signature. An MH maintains a vector with M counters, each counter with $counter\_size$ bits. These $counter\_size$ bits are used to represent an integer value. Initially, the integer value is set to zero. When a data item is inserted into the local cache, a data signature is generated for the data item. If k independent hash functions are used to construct the bloom filter, k bits are set in the data signature. The counters at the position of the bits that are set in the data signature are incremented (Algorithm 4). Likewise, when a data item is evicted from the cache, a data signature is also generated for the data item. The corresponding counters at the position of the bits are decremented (Algorithm 5). By using this proactive signature generation approach, the MH can construct a new cache signature by simply setting the bits at the position of the counters with non-zero values.

Algorithm 4 Inserting a data item into the cache

```
1: procedure InsertDataItem(CounterVector V, DataItem d, int counter_size)
2: // V = {v_1, v_2, ..., v_M};
3: Compute H; // H = {h_1(d), h_2(d), ..., h_k(d)};
4: for all h_i(d) ∈ H do
5:    if v_{h_i(d)} < 2^{counter_size} − 1 then
6:       v_{h_i(d)} ← v_{h_i(d)} + 1;
7:    end if
8: end for
```

The integer value of a counter is bounded by the number of bits used to represent the value, i.e., it is between 0 and $2^{counter\_size} - 1$. Thus, in case that the value of a counter is equal to $2^{counter\_size} - 1$, further increment operation is omitted. Also, if the value is equal to zero, future decrement operation should not be executed. However, there could be overflow and underflow problems in the proactive generation scheme.

Algorithm 5 Removing a data item from the cache

```
1: procedure RemoveDataItem(CounterVector V, DataItem d, int underflow_counter)
2: //V = {v_1, v_2, ..., v_M};
3: Compute H; // H = {h_1(d), h_2(d), ..., h_k(d)};
4: for all h_i(d) ∈ H do
5:    if v_{h_i(d)} = 0 then
6:       underflow_counter ← underflow_counter + 1;
7:    else
8:       v_{h_i(d)} ← v_{h_i(d)} − 1;
9:    end if
10: end for
```

For instance, a single bit is used to represent a counter value, i.e., $counter\_size = 1$, and two data items, d_i and d_j, overlap between their data signatures. The insertion of d_i causes the corresponding counter value to be incremented from zero to one. Then, the insertion of d_j also causes that counter value to be incremented. To avoid an overflow error, the latter increment is discarded. Such avoidance of overflow problem can induce a false negative error. When one of these data items, d_i, is removed from the cache in future, the corresponding counter value is decremented, i.e., the value is changed from one to zero. Then, the cache signature is unable to represent the existence of d_j in the local cache. A peer that queries this cache signature will not be aware of the existence of d_j and its request for d_j will constitute a false negative error.

The analysis of the overflow problem in the proactive signature generation scheme can be referred to as a ball-and-urn problem. In the ball-and-urn model [76], the balls are randomly distributed in the urns, and the number of urns represents the number of counter in the counter vector, whereas the number of balls in an urn represents the value of a counter. The average number of urns with x balls, after β balls are distributed in α urns, can be determined by the equation:

$$\alpha \begin{pmatrix} \beta \\ x \end{pmatrix} \left(\frac{1}{\alpha}\right)^{x} \left(1 - \frac{1}{\alpha}\right)^{\beta - x}. \tag{4.5}$$

Let $Cache$ be a set of cached data items in an MH, denoted $Cache = \{c_1, c_2, \ldots c_{|Cache|}\}$. Hence, the average number of overflowed counters for an MH as derived by Equation 4.5 is:

$$M \begin{pmatrix} k \times |Cache| \\ 2^{counter\_size} \end{pmatrix} \left(\frac{1}{M}\right)^{2^{counter\_size}} \left(1 - \frac{1}{M}\right)^{|Cache|k - 2^{counter\_size}}, \tag{4.6}$$

where $|Cache|$ is the number of cached data items. For instance, let $M = 40000$ and $|Cache| = 100$, Table 4.2 shows the average number of overflowed counters in a counter vector when the number of bits used to represent the counter value varies from one to five with different values of k by using Equation 4.6.

Table 4.2: Average number of overflowed counters. ($M = 40000$, $|Cache| = 100$)

counter_size	**Upper bound**	**Average number of overflowed counters**			
		$k = 1$	$k = 2$	$k = 3$	$k = 4$
1	$2^1 - 1 = 1$	0.1234	0.4950	1.1129	1.9752
2	$2^2 - 1 = 3$	6.1122×10^{-08}	1.0058×10^{-06}	5.1305×10^{-06}	1.6256×10^{-05}
3	$2^3 - 1 = 7$	1.1332×10^{-21}	3.3469×10^{-19}	8.9739×10^{-18}	9.1557×10^{-17}
4	$2^4 - 1 = 15$	1.2508×10^{-51}	1.5681×10^{-46}	1.2663×10^{-43}	1.3972×10^{-41}
5	$2^5 - 1 = 31$	3.0958×10^{-117}	2.5620×10^{-106}	2.7313×10^{-100}	4.2135×10^{-96}

Since the occurrence of counter overflows does not affect the resulted cache signature, the number of overflow errors is not recorded in Algorithm 4. On the other hand, any underflow error directly leads to generate an imprecise cache signature. In Algorithm 5, a counter ($underflow\_counter$) is thus maintained to record the num-

ber of occurrences of an underflow error. When the $underflow\_counter$ reaches an unacceptable level, all counters in the vector will be reset and Algorithm 4 is next applied to each cached data item to re-construct a correct counter vector. Then, $underflow\_counter$ is reset to zero.

4.5 Cache Signature Storage Schemes

There is a tradeoff between the storage space and maintenance overhead for storing cache signatures. When an MH stores the cache signatures of its peers individually, it conserves power as the cache signatures can be handled separately. For instance, when an MH detects a link failure with a peer, it can simply remove the cache signature of that peer from the cache.

On the other hand, if the MH groups the peer signatures together and stores them in a counter vector structure, it enjoys a higher LCH ratio, as this storage method can save on cache space. However, the MH has to suffer from higher maintenance overhead on updating the peer signature. When an MH detects a link failure with a peer, it cannot simply drop the corresponding cache signature from the cache, as it cannot extract the required cache signature from the counter vector structure. To keep the accuracy of its peer signature, the MH has to reset the counter vector structure and re-collect the cache signatures of its peers, and then re-construct the counter vector.

In the proactive generation mechanism, a fixed upper bound of the counter size is adopted, since we can easily find the appropriate counter size to attain an acceptable probability of the occurrence of underflow errors. The MHs can also re-construct the counter vector at a low cost, as the re-construction does not incur any communication between their peers. However, for the cache signature storage scheme, the

re-construction of a counter vector could induce very high overhead because the MHs have to re-collect all required cache signatures from their peers. Therefore, a dynamic upper bound mechanism is adopt to control the counter size, in order to reduce the signature maintenance overhead.

To study the effect of the tradeoff on system performance, we propose three signature storage schemes for storing cache signatures: *individual*, *group* and *hybrid*.

The *individual signature storage scheme* is the most intuitive and simplest one. An MH stores all the received cache signatures of its peers separately. When the MH gets any update information of the cache signatures from peers (the details on the cache signature update will be described in Section 4.6), it updates the cache signatures accordingly. If the MH detects a link failure with a peer, it simply evicts its signature from the cache. This scheme requires the largest cache space to store cache signatures, but it incurs the least maintenance overhead in comparison with the other schemes.

For the *group signature storage scheme*, an MH constructs a counter vector structure to store all cache signatures of its peers. Each MH maintains a vector of M counters, i.e., M is the size of a bloom filter, each counter with $counter\_size$ bits. If an MH has not detected any peer, $counter\_size$ is zero, i.e., the MH needs not to store any peer signature. Once the MH discovers a peer, it creates a counter vector with one bit and set all counters to zero. After the MH grabs the cache signature from the peer, it updates the counter vector by incrementing the counters at the position of the bits that are set in the received cache signature. When the MH is going to increase a counter to a value of $2^{counter\_size}$, $counter\_size$ will be increased by one to avoid an overflow error. Likewise, in case that all counter values fall below $2^{counter\_size-1}$, $counter\_size$ will be decremented to save on cache space. When the MH detects a link failure with a peer whose cache signature is stored in the counter

vector, it resets the counter vector, and then it re-collects all the cache signatures from the remaining neighboring peers. After the MH collects all the required cache signatures, it re-constructs the counter vector. This scheme requires the least cache space to store the cache signatures, but the maintenance overhead incurred by this scheme is much higher than the other schemes.

In the *hybrid signature storage scheme*, an MH makes use of both the individual and group signature storage schemes. The MH first caches all cache signatures of its peers individually. After the MH has cached a cache signature for a period of time, T_{hybrid}, it uses the counter vector to store information of that cache signature and removes it from the cache to save on storage space. When the MH detects a link failure with a peer whose cache signature is stored in the counter vector, the MH resets the counter vector and re-collects the cache signatures from other peers whose cache signatures are stored in the counter vector. It re-constructs the counter vector based on the received cache signatures. On the other hand, if the MH detects a link failure with a peer whose cache signature is stored individually, the MH simply removes the peer's signature from the cache. This scheme saves the cache space by storing some of cache signatures to a counter vector, and reduces communication overheads based on the belief that the cache signatures continuously cached for T_{hybrid} will be cached for a longer time than other cache signatures in the future.

4.6 Cache Signature Exchange Protocol

Cache signature exchange protocol consists of two parts: *new neighbor detection* and *link failure detection*. For the individual signature storage scheme (Algorithms 6 and 7), when an MH has detected a new neighbor through NDP, as described in Chapter 3 (Section 3.4.5), NDP triggers Algorithm 6 to request the cache signature

from the newly detected neighbor. The MH sends a request (*SigRequest*) to the new neighbor, and the neighbor replies the MH with its full cache signature via P2P point-to-point communication. If the free cache space is not enough to cache the received cache signature, the least valuable data item is evicted from the cache, and this process is repeated until there is enough free space to store the cache signature. NDP executes Algorithm 7 when it detects a link failure with a neighbor. It removes the cache signature from the cache.

Algorithm 6 Cache Signature Exchange Protocol for *Individual Signature Storage Scheme* - new neighbor detection at MH m_i

```
1: procedure OnDetectNewNeighbor(Peer P_i, MH m_j)
2: // P_i is a set of identifiers of MHs whose cache signatures are stored in m_i's cache
3: // m_j is a newly discovered neighbor by NDP
4: // Sig_j is the cache signature of m_j
5: if m_j ∉ P_i then
6:     Send SigRequest to m_j;
7:     Receive Sig_j from m_j;
8:     Insert Sig_j into cache;
9:     P_i ← P_i ∪ {m_j};
10: end if
```

Algorithm 7 Cache Signature Exchange Protocol for *Individual Signature Storage Scheme* - link failure detection at MH m_i

```
1: procedure OnDetectLinkFailure(Peer P_i, MH m_k)
2: // P_i is a set of identifiers of MHs whose cache signatures are stored in m_i's cache
3: // a link failure with an MH m_k is detected by NDP
4: // Sig_k is the cache signature of m_k
5: if m_k ∈ P_i then
6:     P_i ← P_i − {m_k};
7:     Remove Sig_k from cache;
8: end if
```

For the group signature storage scheme (Algorithms 8 and 9), when an MH has detected a new neighbor through NDP, it sends a *SigRequest* to the newly discovered neighbor to request its cache signature, and the neighbor then replies the MH with

its full cache signature via P2P point-to-point communication. Algorithm 8 updates the counter vector by incrementing the counters at the position of the bits that are set in the received cache signature. NDP executes Algorithm 9 when it detects a link failure with a neighbor. It resets the counter vector and *counter_size*, and then it re-collects the cache signatures of other peers.

Algorithm 8 Cache Signature Exchange Protocol for *Group Signature Storage Scheme* - new neighbor detection at MH m_i

```
1: procedure OnDetectNewNeighbor(CounterVector V_i, Peer P_i^G, MH m_j)
2: // V_i = {v_1, v_2, ..., v_M}
3: // P_i^G is a set of identifiers of MHs whose cache signatures are stored in a counter
   vector in m_i's cache
4: // m_j is a newly discovered neighbor by NDP
5: // Sig_j is the cache signature of m_j, Sig_j = {s_1, s_2, ..., s_M}
6: if m_j ∉ P_i^G then
7:    Send SigRequest to m_j;
8:    Receive Sig_j from m_j;
9:    P_i^G ← P_i^G ∪ {m_j};
10:   for all s_h ∈ Sig_j do
11:      if s_h = 1 then
12:         if v_h = 2^counter_size_i − 1 then
13:            counter_size_i ← counter_size_i + 1;
14:         end if
15:         v_h ← v_h + 1;
16:      end if
17:   end for
18: end if
```

For the hybrid signature storage scheme (Algorithms 10, 11 and 12), when an MH has detected a new neighbor through NDP, it requests the cache signature from the newly detected neighbor by sending a (*SigRequest*) request to the new neighbor, and the neighbor replies the MH with its full cache signature via P2P point-to-point communication. The MH stores the received cache signature individually in the cache. NDP triggers the execution of Algorithm 11 when it detects a link failure with a peer. If the cache signature of the peer is stored in the counter vector, the MH resets the

Algorithm 9 Cache Signature Exchange Protocol for *Group Signature Storage Scheme* - link failure detection at MH m_i

```
1: procedure OnDetectLinkFailure(CounterVector V_i, Peer P_i^G, MH m_k)
2: // V_i = {v_1, v_2, ..., v_M}
3: // P_i^G is a set of identifiers of MHs whose cache signatures are stored in a counter
   vector in m_i's cache
4: // a link failure with a MH m_k is detected by NDP
5: if m_k ∈ P_i^G then
6:    for all v_g ∈ V_i do
7:       v_g ← 0;
8:    end for
9:    P_i^G ← P_i^G − {m_k};
10:   counter_size_i ← 0;
11:   // re-collect cache signatures from all remaining members in P_i^G;
12:   for all m_j ∈ P_i^G do
13:      Send SigRequest to m_j;
14:      Receive Sig_j from m_j;
15:      for all s_h ∈ Sig_j do
16:         if s_h = 1 then
17:            if v_h = 2^counter_size_i − 1 then
18:               counter_size_i ← counter_size_i + 1;
19:            end if
20:            v_h ← v_h + 1;
21:         end if
22:      end for
23:   end for
24: end if
```

counter vector and *counter_size*, and then it re-collects the cache signatures from the remaining peers whose cache signatures are stored in the counter vector. Otherwise, the MH simply drops the cache signature from the cache. In addition, Algorithm 12 is periodically executed to decide whether any cache signature should be stored in the counter vector. It stores a cache signature in the counter vector if the MH has cached the cache signature for a period time T_{hybrid}. To store an individual cache signature in the counter vector, the MH updates the counter vector by incrementing the counters at the position of the bits which are set in the cache signature. Then, the cache signature is removed from the cache to save on cache space.

Algorithm 10 Cache Signature Exchange Protocol for *Hybrid Signature Storage Scheme* - new neighbor detection at MH m_i

```
1: procedure OnDetectNewNeighbor(Peer P_i, MH m_j)
2: // P_i is a set of identifiers of MHs whose cache signatures are stored in m_i's cache
3: // m_j is a newly discovered neighbor by NDP
4: // Sig_j is the cache signature of m_j
5: if m_j ∉ P_i then
6:    Send SigRequest to m_j;
7:    Receive Sig_j from m_j;
8:    P_i ← P_i ∪ {m_j};
9:    // now() returns the current time
10:   signature_receive_ts_j ← now();
11: end if
```

To reduce communication overhead on updating cache signature, the MHs embed the signature update information to the request message that is broadcast to all their neighboring peers. When a peer receives the request message along with signature update information, it extracts the signature update information from the request message, and then it updates either the cache signature or the counter vector accordingly. If an MH has no neighbor at all, it does not piggyback any signature update information into the request message. The signature update information is maintained in two lists: one list storing the bit position that has been set, and another one storing the bit position that has been reset since the last time the MH broadcasts a request message along with signature update information. All bit positions are stored as integer values. If a bit position exists in both lists, the change is annihilated and the bit position is removed from them. After the MH broadcasts signature update information to the peers, it reset both lists.

To ensure the accuracy of cache signatures, an MH that has not generated any request to its neighboring peers for a period of time T_{sig} is required to broadcast its signature update information to its neighbors, unless there is no change in its cache signature.

Algorithm 11 Cache Signature Exchange Protocol for *Hybrid Signature Storage Scheme* - link failure detection at MH m_i

```
1: procedure OnDetectLinkFailure(CounterVector V_i, Peer P_i, Peer P_i^G, MH m_k)
2: // V_i = {v_1, v_2, ..., v_M}
3: // P_i is a set of identifiers of MHs whose cache signatures are stored in m_i's cache, not the counter vector
4: // P_i^G is a set of identifiers of MHs whose cache signatures are stored in a counter vector in m_i's cache
5: // a link failure with an MH m_k is detected by NDP
6: if m_k ∈ P_i^G then
7:    for all v_g ∈ V_i do
8:       v_g ← 0;
9:    end for
10:   P_i^G ← P_i^G − {m_k};
11:   counter_size_i ← 0;
12:   // re-collect cache signatures from all remaining members in P_i^G;
13:   for all m_j ∈ P_i^G do
14:      Send SigRequest to m_j;
15:      Receive Sig_j from m_j;
16:      for all s_h ∈ Sig_j do
17:         if s_h = 1 then
18:            if v_h = 2^counter_size_i − 1 then
19:               counter_size_i ← counter_size_i + 1;
20:            end if
21:            v_h ← v_h + 1;
22:         end if
23:      end for
24:   end for
25: else if m_k ∈ P_i then
26:   P_i ← P_i − {m_k};
27:   Remove Sig_k from cache;
28: end if
```

When an MH is disconnected from the network either voluntarily or involuntarily, it will miss some of the signature update information sent by its peers. After the MH re-connects to the network, some cache signatures may become stale. The system performance could be degraded, as the peer signature cannot provide precise information for the MH to decide whether to search for the desired data item in the Peer Cache layer when it encounters a local cache miss. Therefore, the cache signature

Algorithm 12 Cache Signature Exchange Protocol for *Hybrid Signature Storage Scheme* - signature time stamp checking at MH m_i

```
1: procedure CheckSignTimeStamp(CounterVector V_i, Peer P_i, Peer P_i^G)
2: // V_i = {v_1, v_2, ..., v_M}
3: // P_i is a set of identifiers of MHs whose cache signatures are stored in m_i's cache,
   not the counter vector
4: // P_i^G is a set of identifiers of MHs whose cache signatures are stored in a counter
   vector in m_i's cache
5: for all m_j ∈ P_i do
6:   if now() − signature_receive_ts_j ≥ T_hybrid then
7:     P_i ← P_i − {m_j};
8:     P_i^G ← P_i^G ∪ {m_j};
9:     // Sig_j is a cache signature of m_j
10:    for all s_h ∈ Sig_j do
11:      if s_h = 1 then
12:        if v_h = 2^counter_size_i − 1 then
13:          counter_size_i ← counter_size_i + 1;
14:        end if
15:        v_h ← v_h + 1;
16:      end if
17:    end for
18:    Remove Sig_k from cache;
19:  end if
20: end for
```

exchange protocol is extended to handle client disconnection to preserve the accuracy of the bypassing mechanism.

When an MH broadcasts a request to its peers with the signature update information, it records the time, *last_signature_update_ts*. The MH immediately records the time, when it is disconnected from the network, *disconnection_ts*. After the MH reconnects to the network, it broadcasts a request to its neighbors with *disconnection_ts*. Its neighbors compare its *last_signature_update_ts* with *disconnection_ts*. For a peer, if *last_signature_update_ts* $\geq$ *disconnection_ts*, the cached cache signature of this peer is stale, and the peer turns in its full cache signature to the requesting MH. Otherwise, the cache signature is still valid, the peer sends an acknowledgement message to the MH. After the MH receives response from all the peers, the MH performs the

signature update procedure based on the adopted signature storage scheme.

For the individual signature storage scheme, the MH simply replaces the stale cache signatures with the newly received ones. For the group signature storage scheme, if there is any stale cache signature, the MH needs to reset the counter vector and re-collect the cache signatures from the peers that have responded with acknowledgement messages. Then, the MH re-constructs the counter vector based on the collected cache signatures. For the hybrid signature storage scheme, if the stale cache signature is not stored in the counter vector, the MH can simply remove them with received cache signatures. Otherwise, the MH has to reset the counter vector and re-collect the cache signatures from the peers that have responded with acknowledgement messages and their cache signatures are stored in the counter vector. The MH then re-constructs the counter vector based on the received cache signatures.

4.7 Simulation Model

The simulation model of COCA with cache signature scheme is based on the model defined in Chapter 3 with additional parameter settings for the cache signature scheme, as shown in Table 4.3.

4.8 Simulation Results

We adopt the same simulation model, including power consumption model, mobility model, data access pattern, server model and network model, as defined in Chapter 3 to compare the performance of COCA with cache signature scheme (denoted as CC-SIG) with a conventional caching scheme that does not involve any cooperation among MHs (denoted as NC) and standard COCA (denoted as CC) in the pull-based, push-based and hybrid environments. All schemes use LRU cache replacement policy.

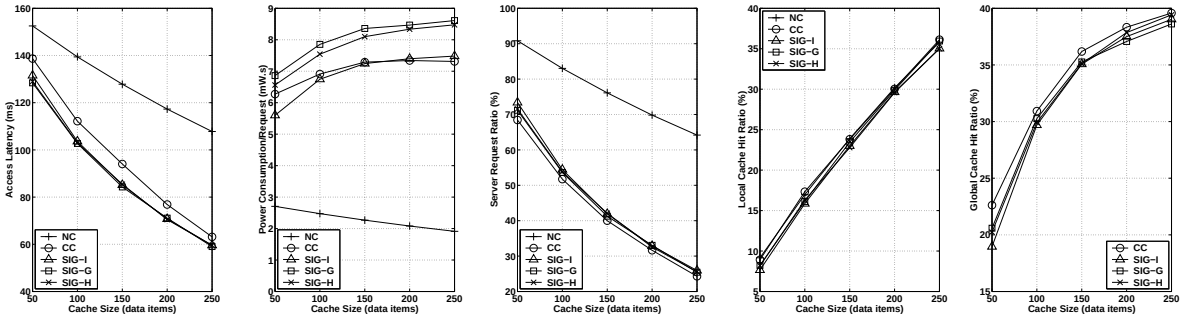
Table 4.3: Simulation parameters and default settings for COCA with cache signature scheme.

Parameter	Description	Default Value
M	Signature length	40000
k	No. of independent hash functions	2
R	Maximum run-length	511
$counter\_size$	Counter size of a counter vector used in proactive generation of a cache signature	1
T_{sig}	Time period of broadcasting signature update information to neighboring peers	10 s
T_{hybrid}	Time period of storing a cache signature in a counter vector	10 s

As the simulation result presented in Chapter 3 exhibits that the flat disk broadcasting scheduling always performs worse than the broadcast disk broadcast scheduling, we only consider the broadcast disk [2] in the push-based environment. For the hybrid environment, the simulation result depicts that the trend of the three broadcast channel allocation policies are similar to each other, except two series of experiments: common hot spot and number of MHs. For simplicity, we only consider the policy of $PushChannel = 50\%$ in the hybrid environment. To study the tradeoff between storage space and maintenance overheard for storing cache signatures, three proposed cache signature schemes, individual, group and hybrid are applied to CC-SIG (denoted as SIG-I, SIG-G and SIG-H respectively).

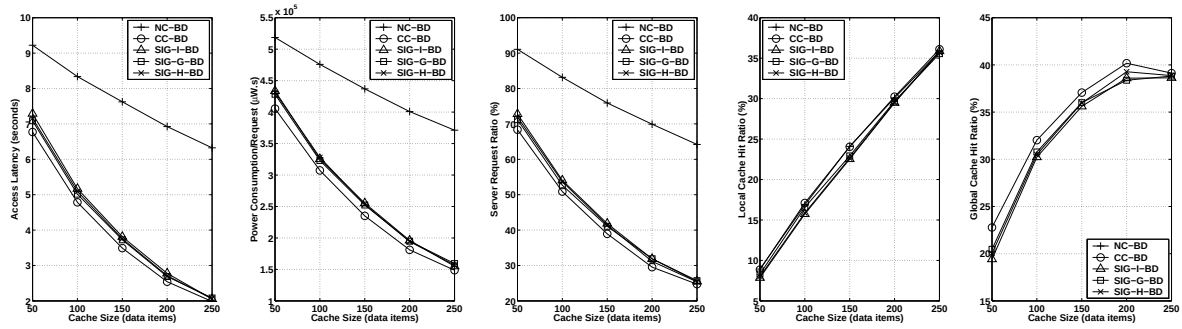
A series of simulated experiments is conducted by varying several parameters: cache size, data item size, access patterns, client disconnection probability, mobility speed, number of MHs in the system. The performance metrics include access latency, power consumption, server request ratio, LCH ratio and GCH ratio.

4.8.1 Effect of Cache Size



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.3: Effect of cache size in a pure pull-based environment.

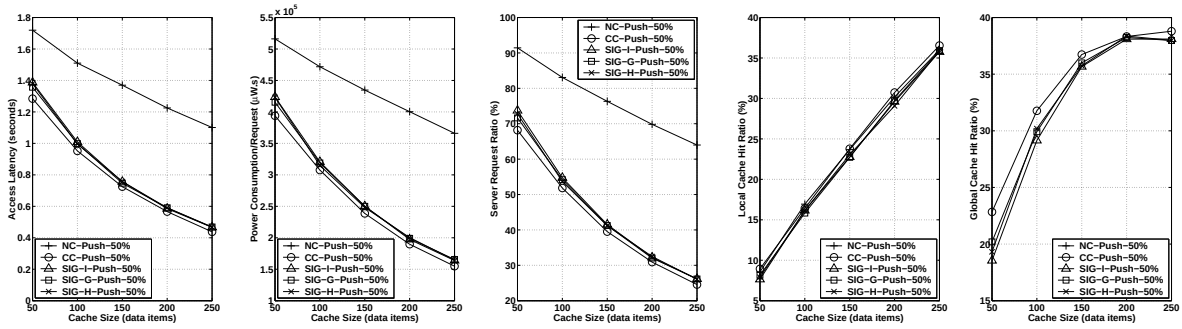


(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.4: Effect of cache size in a pure push-based environment.

Our first experiment studies the effect of cache size on system performance by varying the cache size from 50 to 250 data items.

For the pure pull-based environment, Figures 4.3(a) and 4.3(c) show that all schemes exhibit better access latency and server request ratio with increasing cache size. This is because the LCH ratio increases as the cache size gets larger, as depicted in Figure 4.3(d). In terms of access latency and server request ratio, CC and CC-SIG outperform NC. For an MH adopting CC and CC-SIG, other than achieving a higher LCH ratio as the cache size gets larger, it also enjoys a higher GCH ratio, as shown in Figure 4.3(e), because the chance of some neighboring peers caching the required



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.5: Effect of cache size in a hybrid environment.

data items increases with the larger cache size. CC-SIG gives the best performance in access latency, since the cache signature scheme effectively provides hints for the MHs to bypass the Peer Cache layer. Therefore, the MHs adopting CC-SIG does not need to waste time to search the Peer Cache layer, as they are likely to encounter global cache misses.

In NC, the MHs witness a larger LCH ratio as the cache size gets larger, so the power consumption is reduced, as shown in Figure 4.3(b). It also suggests that the cost of adopting CC and CC-SIG is higher power consumption. When the MHs enjoy a higher LCH ratio, they can reduce the power consumption in searching the Peer Cache layer and retrieving the required data items from the MSS. However, when the GCH ratio increases, more peers cache the required data items and need to forward them to the requesting MHs. Therefore, the MHs have to consume more power for sending data items and discarding unintended messages over P2P communication channels as they are residing in the transmission range of the source MH, destination MH or both.

Among CC-SIG, SIG-G gives the best performance in access latency, but the MHs adopting SIG-G have to consume more power than the other cache signature storage schemes to re-collect all the cache signatures of their peers when they detect any link

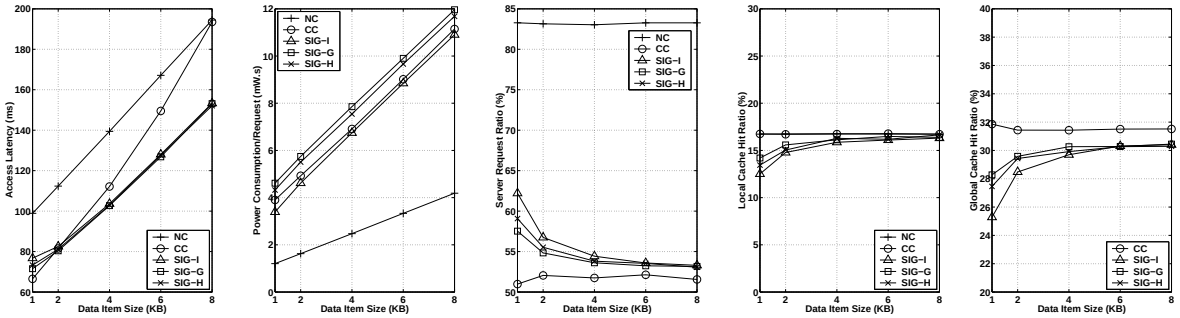
failure with their peers, as shown in Figures 4.3(a) and 4.3(b). For SIG-H, it gives the second best performance in access latency, but the MHs adopting this scheme also consume more power than SIG-I. SIG-G and SIG-H slightly improve access latency, but they incur much more power consumption than SIG-I. Thus, we suggest that the MH should adopt SIG-I in the pull-based mobile environment rather than the other two schemes.

In the push-based environment, CC-SIG cannot improve access latency, as shown in Figure 4.4(a). This is because CC-SIG is a filtering mechanism for an MH to determine whether to search the Peer Cache layer. When the peer signature of an MH indicates that no peer is likely caching its desired data item, the MH merely listens to the broadcast channel and overlooks the Peer Cache layer. As the MH can start to tune in to the broadcast channel and search the Peer Cache Layer at the same time after it encounters a local cache miss, so CC-SIG cannot improve access latency. The MHs adopting CC-SIG can bypass the Peer Cache layer when the peer signature indicates that the required data items are not in the global cache, so they can reduce power consumption compared with CC, in which the MHs have to search the Peer Cache layer for every local cache miss. However, the storage of cache signatures reduce the effective cache space of an MH. As a result, the MH suffers from a lower LCH ratio, as depicted in Figure 4.4(d). The MH has to consume more power to obtain the required data items either from the broadcast channel or the cache of its peers and exchange its cache signature and signature update information with its peers. All these communication overheads offset the benefit of the CC-SIG schemes.

In the hybrid environment, we can draw similar conclusions as in the push-based environment, as exhibited in Figure 4.5. CC-SIG cannot improve system performance, as the effective cache size is the dominant factors in access latency and power consumption. In addition, the exchange of cache signatures and signature update infor-

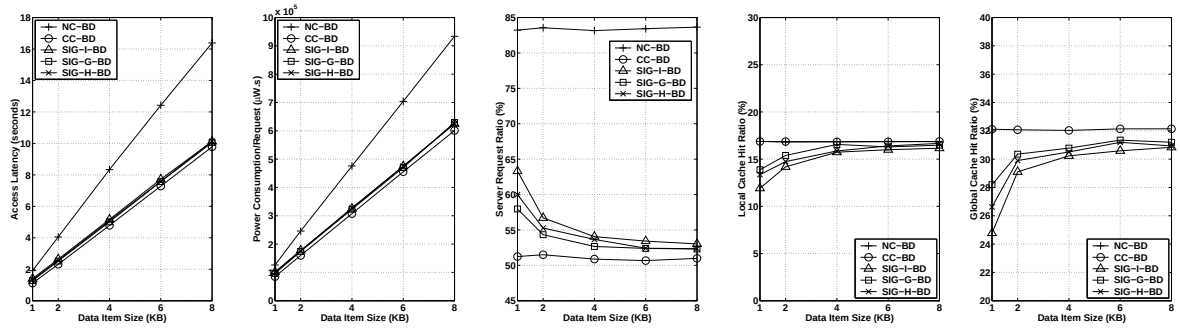
mation among MHs accounts for higher power consumption in CC-SIG than CC.

4.8.2 Effect of Data Item Size



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.6: Effect of data item size in a pure pull-based environment.



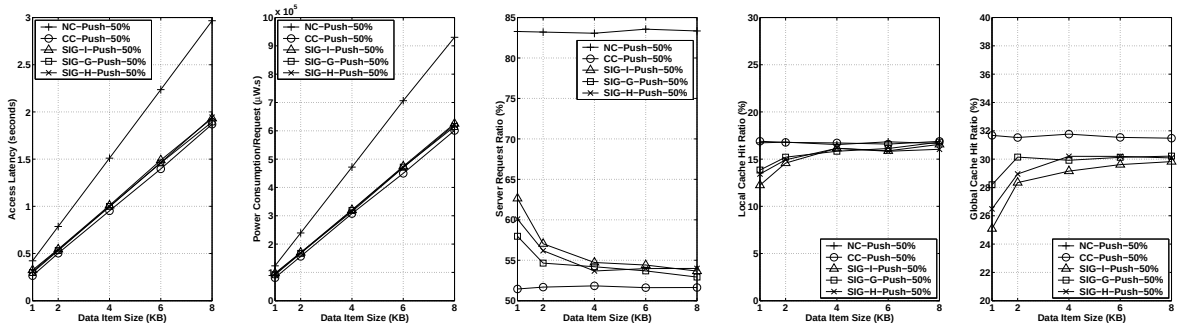
(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.7: Effect of data item size in a pure push-based environment.

This series of experiment studies the effect of data item size on system performance by varying the data item size from one to eight kilobytes (KB).

Figures 4.6, 4.7 and 4.8 show that the access latency and power consumption increase, as the data item size gets larger. This is because the data item with larger size requires longer transmission time that induces higher power consumption.

In the pull-based environment, the MHs adopting CC-SIG suffer from a lower GCH ratio than CC as depicted in Figure 4.6(e). When the data item size gets larger,



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.8: Effect of data item size in a hybrid environment.

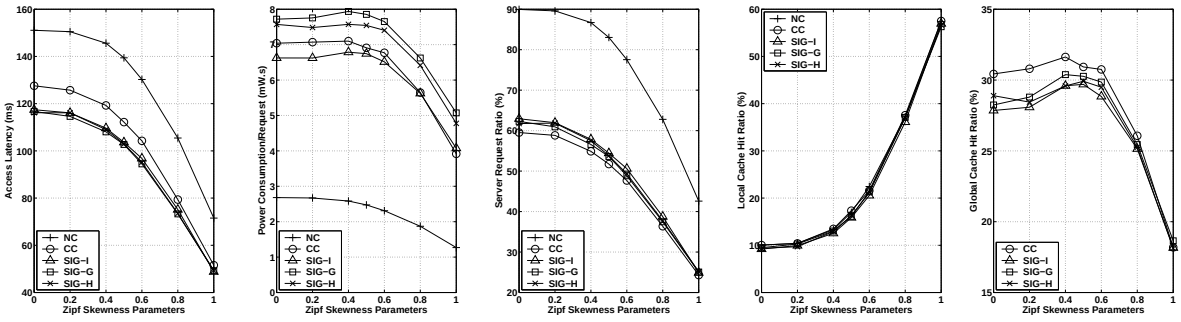
the GCH ratio of CC-SIG increases, as the impact of data item size on CC-SIG is much smaller than on CC, as shown in Figure 4.6(a). When the data item size is small, i.e., less than or equal to 2 KB, CC-SIG is the worst performer in the access latency. However, CC-SIG performs better than CC and NC, as the data item size further increases. It is due to fact that the overhead of cache space occupied by the cache signature as reflected by the ratio of the cache signature size to the total cache size reduces with increasing data item size. When the data item size is small, this overhead ratio is higher and the impact of the storage overhead significantly degrades system performance, so it hurts the LCH and GCH ratios, as depicted in Figures 4.6(d) and 4.6(e). As the data item size gets larger, this ratio decreases, i.e., the impact of the storage overhead on the cache signature reduces. Thus, the LCH and GCH ratios rise with increasing data item size.

When the data item size increases, the MHs adopting NC have to consume more power on receiving data items from the MSS. In COCA schemes, the MHs not only receive data items from the MSS, but they also consume power on forwarding required data items to other peers. As a result, the MHs adopting COCA schemes are expected to consume more power than NC with increasing data item size, as shown in Figure 4.6(b).

For CC-SIG, when the data item size is small, i.e., 1 KB, the tradeoff between storage space and maintenance overhead is obvious. SIG-G performs better than the other two schemes in terms of access latency, as depicted in Figure 4.6(a), as it can improve the effective cache space, so the MHs adopting SIG-G achieve higher LCH and GCH ratios, as shown in Figures 4.6(d) and 4.6(e). However, they have to consume more power than the other two schemes, because they need to re-collect the cache signatures from their peers when they detect any link failure with their peers. SIG-I gives the longest access latency than SIG-G and SIG-H, but it consumes less power in return. The tradeoff becomes negligible with increasing data item size because the overhead of storing cache signatures reduces.

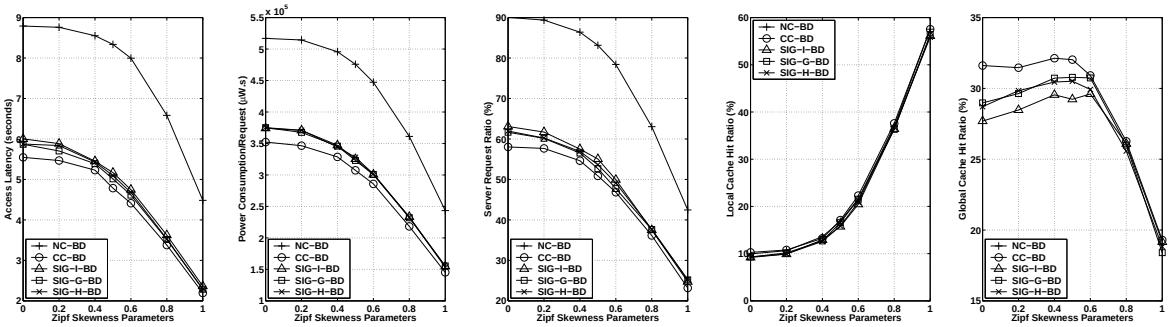
In the push-based and hybrid environments, the effective cache size is the most important factor to system performance in these environments. The effective cache size of CC is better than CC-SIG because the CC-SIG schemes use a portion of cache space to store cache signatures. Thus, CC-SIG performs worse than CC, similar to the result in Section 4.8.1.

4.8.3 Effect of Skewness in Access Pattern



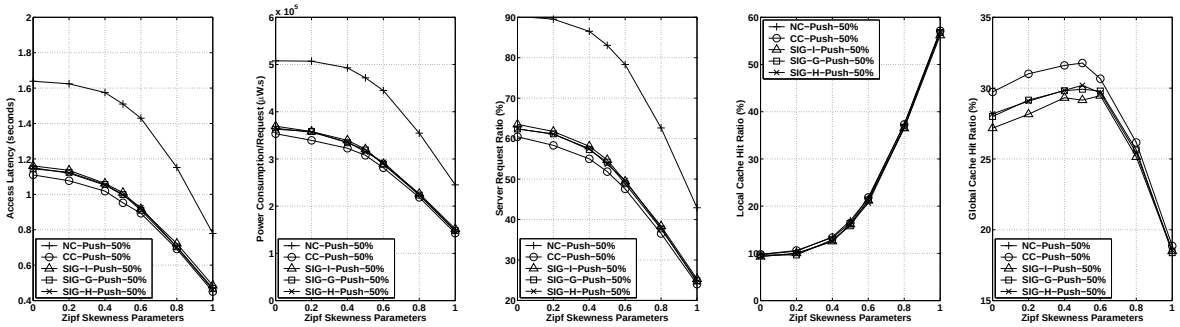
(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.9: Effect of skewness in access pattern in a pure pull-based environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.10: Effect of skewness in access pattern in a pure push-based environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.11: Effect of skewness in access pattern in a hybrid environment.

In this section, we study the effect of skewness in access pattern on system performance by varying the Zipfian parameter value from zero to one.

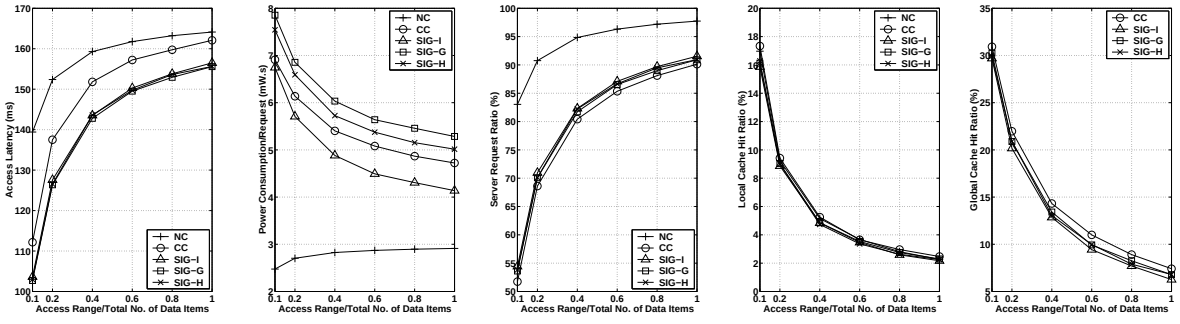
The result shows that all schemes give the worst performance when the skewness parameter is equal to zero. This is because the MHs uniformly access data items within their access range. The system performance improves with increasing skewness parameter value. As the access pattern becomes more skewed, more required data items can be obtained in the local cache, thereby increasing the LCH ratio, as depicted in Figure 4.9(d). When the MHs record higher a LCH ratio with increasing skewness parameter value, the access latency and power consumption improve as a result. Figures 4.9(a) and 4.9(c) illustrate that COCA schemes outperform NC in terms of

access latency and server request ratio.

SIG-G gives the shortest access latency, as shown in Figurer 4.9(a), because the MHs adopting SIG-G use less cache space to store the peer signature than SIG-I and SIG-H, but they consume more battery power to re-construct the counter vector when they detect any link failure with their peers. SIG-I performs the worst in access latency, but it can conserve more power than the other CC-SIG schemes and CC. Although the access latency of SIG-I is slightly worse than SIG-G and SIG-H, the MHs can take advantage of conserving battery power from SIG-I. When the remaining battery power on an MH is low, SIG-I is thus more preferable than the other two schemes.

In the push-based and hybrid environments, CC-SIG performs worst than CC in terms of all performance metrics because the effective cache space is a key factor to system performance. As the MHs adopting CC-SIG use a portion of cache space to store peer signatures, the effective cache space is reduced and they suffer from lower LCH and GCH ratios than CC, as depicted in Figures 4.10(d) and 4.10(e) for the push-based environment, and Figures 4.11(d) and 4.11(e) for the hybrid environment.

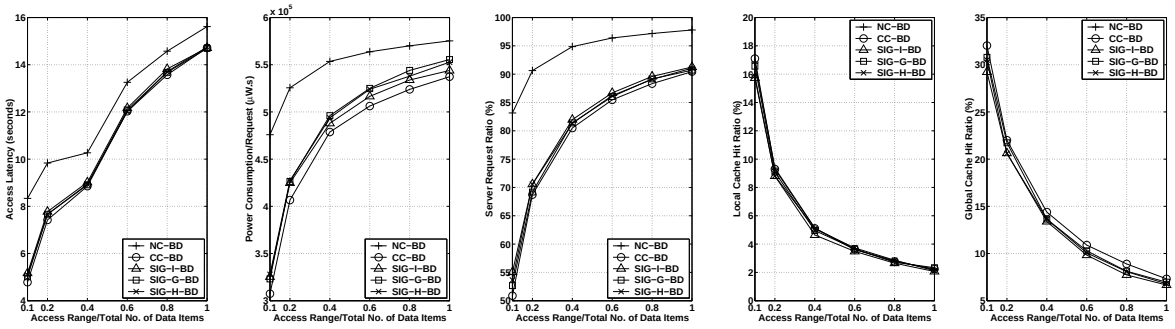
4.8.4 Effect of Access Density



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

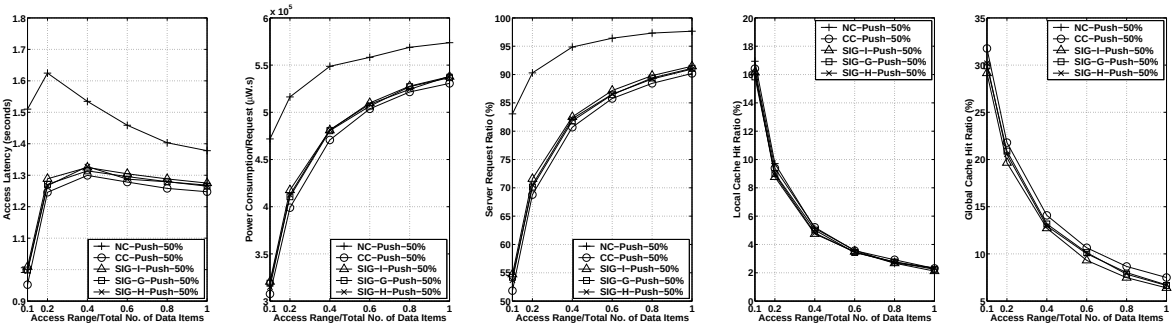
Figure 4.12: Effect of access density in a pure pull-based environment.

In this series of experiment, we study the effect of access density on system per-



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.13: Effect of access density in a pure push-based environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.14: Effect of access density in a hybrid environment.

formance by changing the access range from 1000 to 10000.

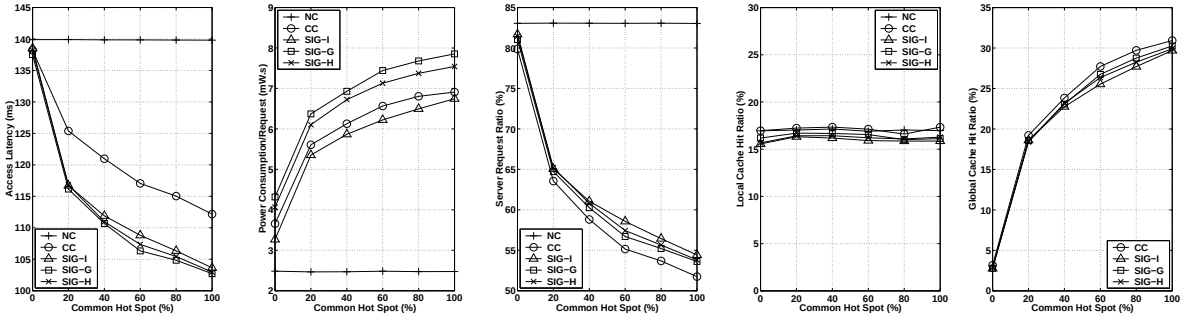
Figures 4.12(a) and 4.12(c) show that CC and CC-SIG perform better than NC, and CC-SIG is the best performer in terms of access latency and server request ratio in the pull-based environment. The access latency and server request ratio increase when the access density gets lower. The lower the access density ratio, the more distinct data items an MH will access; thus, it reduces the LCH ratio as shown in Figure 4.12(d). Likewise, the GCH ratio drops, when the access density gets lower, as depicted in Figure 4.12(d). When the MHs are interested in a large number of distinct data items, the probability of some peers caching the required data items is reduced.

The power consumption of NC increases with decreasing access density because

there is a higher probability for the MHs to enlist the server for help, i.e., higher server request ratio, as shown in Figure 4.12(b). On the contrary, the power consumption of CC and CC-SIG is reduced, as the access density gets lower, due to the fact that power consumption of a global cache access is higher than a server access. Thus, when the GCH ratio of CC and CC-SIG reduces as the access density gets lower, the power consumption also drops. The cache signature scheme can further reduce the power consumption, as the MHs can save power by reducing the chance of having to exhaustively search the Peer Cache layer for every local cache miss.

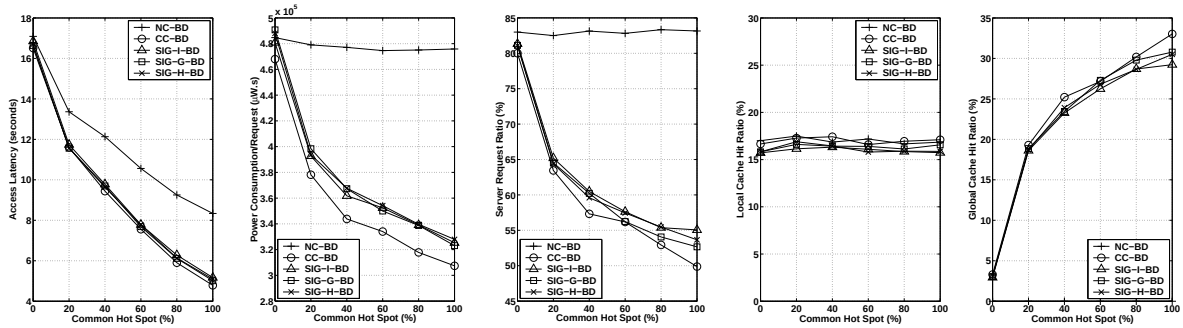
Figure 4.12(a) shows that SIG-G gives the best performance in access latency, as it allows the MHs to reduce the overhead of storing cache signatures. However, there is a tradeoff between the storage space and maintenance overhead. Although SIG-G reduces the storage overhead by storing all cache signatures in a counter vector, it increases maintenance overhead for the MHs, as they have to re-build the counter vector when they detect any link failure with their peers. Therefore, SIG-G incurs higher power consumption than SIG-I and SIG-H. The access latency of SIG-I is larger than the other two CC-SIG schemes, but it effectively reduces power consumption in comparison with the other CC-SIG schemes and CC, as illustrated in Figure 4.12(b).

However, CC-SIG incurs higher access latency and power consumption than CC in the push-based and hybrid environments, as shown in Figures 4.13 and 4.14 respectively, because the CC-SIG scheme reduce the effective cache space for storing cache signatures. As a result, there is a higher chance for the MHs adopting CC-SIG to obtain their desired data items from the MSS, as they suffer from lower LCH and GCH ratios than CC, as depicted in Figures 4.13(d) and 4.13(e) for the push-based environment, and Figures 4.14(d) and 4.14(e) for the hybrid environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.15: Effect of common hot spot in a pure pull-based environment.



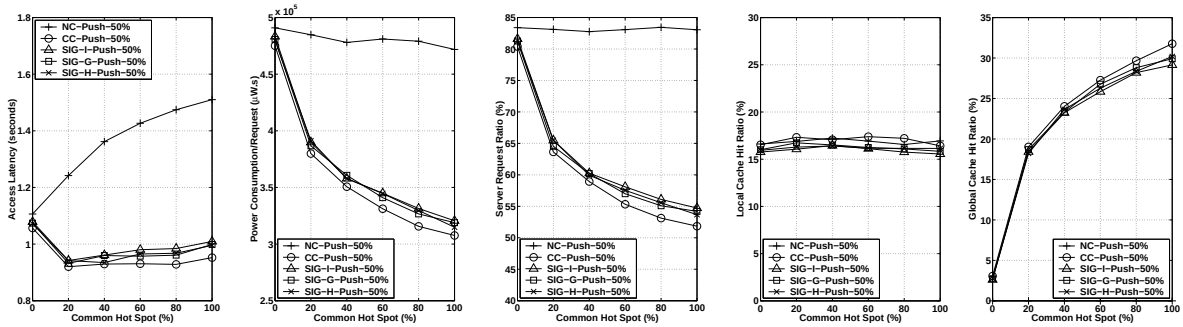
(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.16: Effect of common hot spot in a pure push-based environment.

4.8.5 Effect of Common Hot Spot

We next study the effect of common hot spot of all MHs by varying the percentage of common hot spot from zero to 100 percent. The MHs possess random access range, when the percentage of common hot spot is zero. On the other hand, they own the same access range, as the percentage of common hot spot is 100 percent.

The simulation result shows that the performance of NC is not affected by increasing the percentage of common hot spot among MHs in pure pull-based environment, since there is no cooperation among MHs in NC. The MHs adopting CC and CC-SIG record better access latency and server request ratio, as depicted in Figures 4.15(a) and 4.15(c), because they exhibit more common data access similarity. In COCA,



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.17: Effect of common hot spot in a hybrid environment.

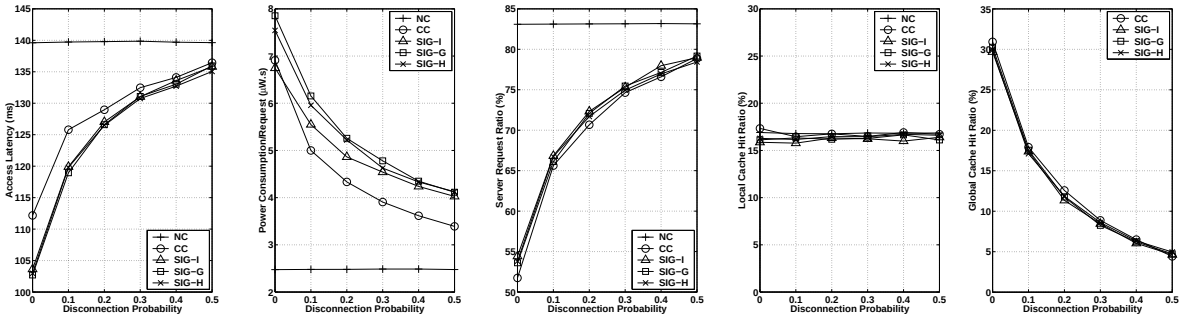
if the MHs possess similar access pattern, they have a higher chance to obtain required data items from their peers, resulting in a higher GCH ratio, as shown in Figure 4.15(e). However, the side effect of a higher GCH ratio is that the MHs have to consume more power to turn in the required data items to the requesting peers. The power consumption increases, as the MHs share more common access patterns, as depicted in Figure 4.15(b).

In terms of access latency and power consumption, CC-SIG performs better than CC. However, CC-SIG incurs higher server request ratio than CC. This is because the effective cache size of the MHs adopting CC-SIG is less than that with CC, as certain portion of local cache space is used for storing their peers' cache signatures. Figure 4.15(b) exhibits that the power consumption of SIG-I is better than CC, as the MHs adopting SIG-I suffer from a lower GCH ratio, and they are able to bypass the Peer Cache layer to save on power. Also, the power consumption of SIG-I is less than the other CC-SIG schemes because the MHs enjoy a lower maintenance overhead on the stored cache signatures. However, the drawback of SIG-I is a reduction in effective cache size. As a result, it records a slightly longer access latency than SIG-G and SIG-H, as shown in Figure 4.15(a).

In the push-based and hybrid environments, CC is the best performer in access

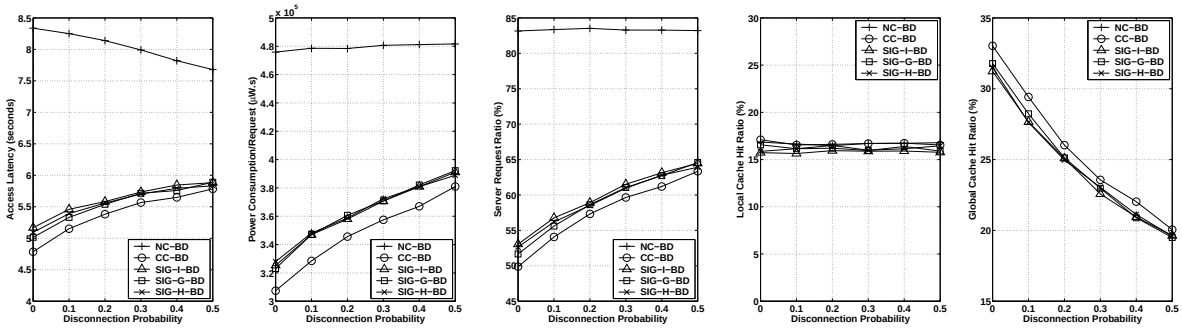
latency, power consumption and server request ratio, as depicted in Figures 4.16 and 4.17. The performance of CC-SIG is worse than CC because the CC-SIG schemes suffer from a lower effective cache size. As there is a higher chance for the MHs adopting CC-SIG to encounter local and global cache misses, they have to spend more time and consume more power to obtain their desired data items from the MSS.

4.8.6 Effect of Client Disconnection Probability



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

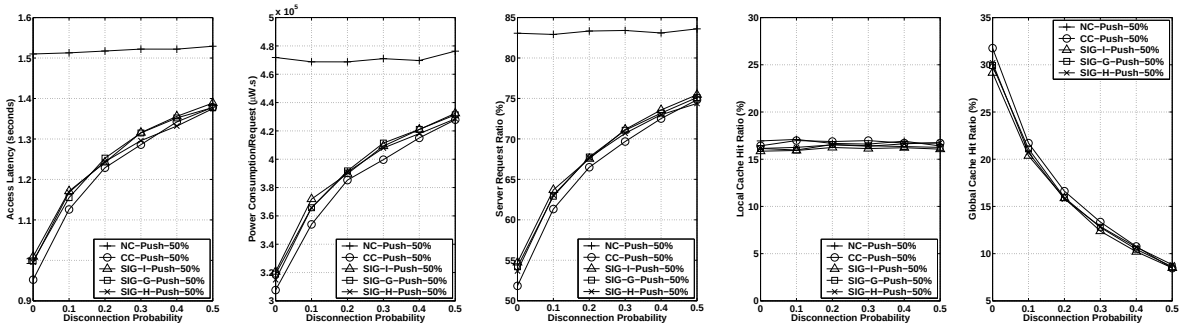
Figure 4.18: Effect of client disconnection probability in a pure pull-based environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.19: Effect of client disconnection probability in a pure push-based environment.

We also study the effect of client disconnection probability on system performance by increasing the disconnection probability that is from 0 to 0.5.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.20: Effect of client disconnection probability in a hybrid environment.

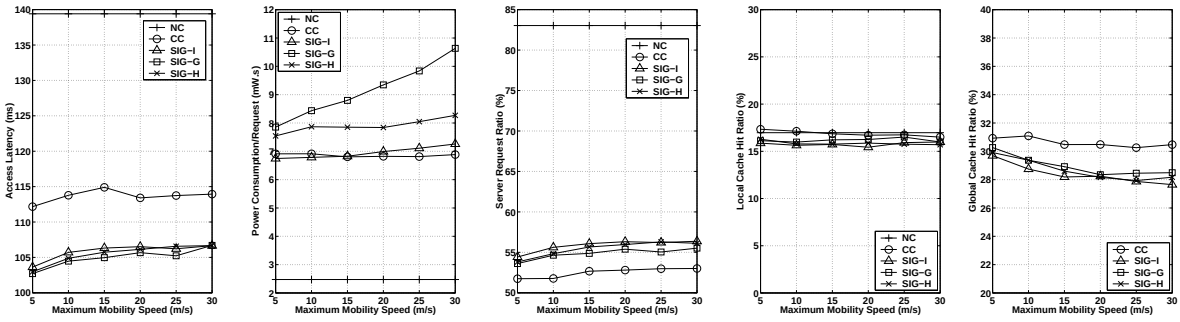
In the pull-based environment, there is no cooperation among peers in NC, the MHs adopting NC have not been affected by varying the disconnection probability. However, the performance of COCA schemes is more sensitive to the disconnection probability. Figures 4.18(a) and 4.18(c) show that the access latency and server request ratio increase, as the client disconnection probability gets higher. The higher the client disconnection probability, the less the peers can help the requesting MHs, so that the GCH ratio degrades with increasing the client disconnection probability, as depicted in Figure 4.18(e).

When an MH disconnects itself from the network, it cannot handle the requests issued from other peers, and need not discard any unintended messages. With a higher client disconnection probability, the power consumption reduces, as shown in Figure 4.18(b). It also shows that CC-SIG incurs higher power consumption than CC with increasing client disconnection probability. In SIG-I, after the MHs re-connect to the network, they have to validate the cached peer signatures with their peers, and need to replace the stale cache signatures with the up-to-date cache signature of relevant peers, so they consume more power on signature validation. For SIG-G and SIG-H, if any cache signature that is stored in a counter vector is stale, the MHs have to re-collect all cache signatures of their peers and re-build the counter vector

structure. Thus, the power consumption of SIG-G and SIG-H is higher than SIG-I.

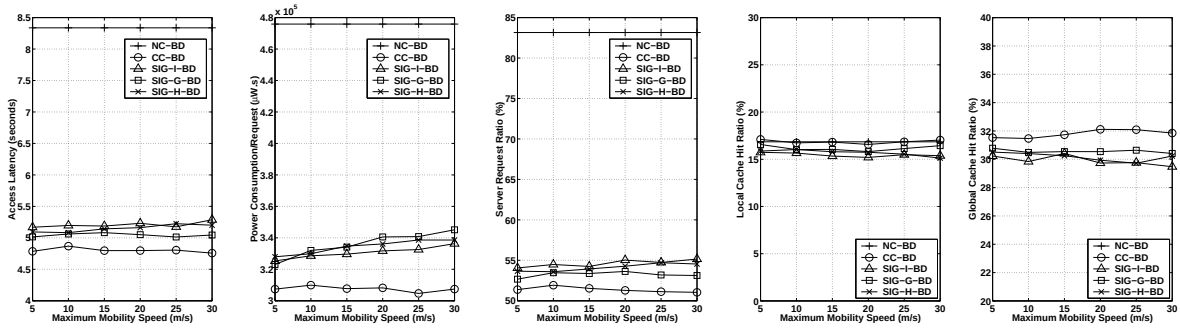
In the push-based and hybrid environments, CC gives the best performance, as exhibited in Figures 4.19 and 4.20. CC-SIG performs worse than CC, also due to the fact that the MHs adopting the CC-SIG scheme suffer from a lower effective cache size than those with CC. Therefore, the CC-SIG MHs have to spend more time and consume more power to retrieve their required data items from the MSS.

4.8.7 Effect of Mobility Speed



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

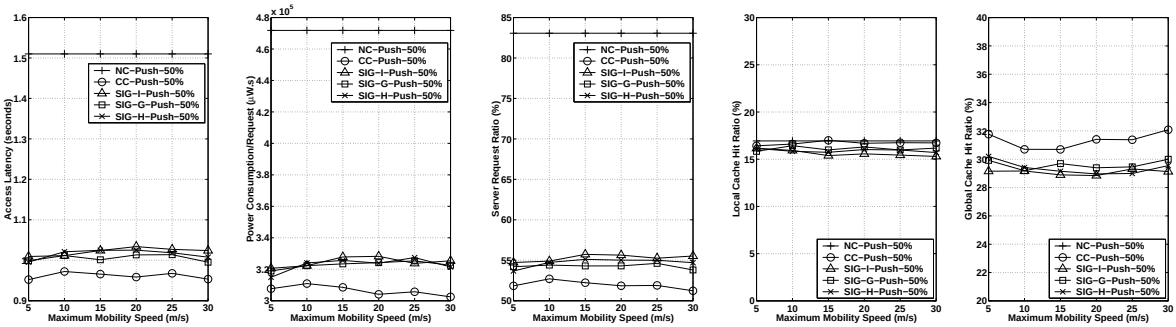
Figure 4.21: Effect of mobility speed in a pure pull-based environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.22: Effect of mobility speed in a pure push-based environment.

In this experiment, we study the effect of mobility speed on system performance by increasing the maximum mobility speed from 5 m/s to 30 m/s.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.23: Effect of mobility speed in a hybrid environment.

In the pull-based environment, the performance of CC is slightly affected by varying the movement speed. However, CC-SIG is more sensitive to the movement speed than CC. When the mobility of the MHs accelerates, the GCH ratio of CC-SIG gets lower. With the client moving with high speed, the neighborhood membership of the MHs changes frequently. The frequent changes in neighborhood membership in an environment with high client movement speed increases the network traffic and degrades the precision of the filtering mechanism.

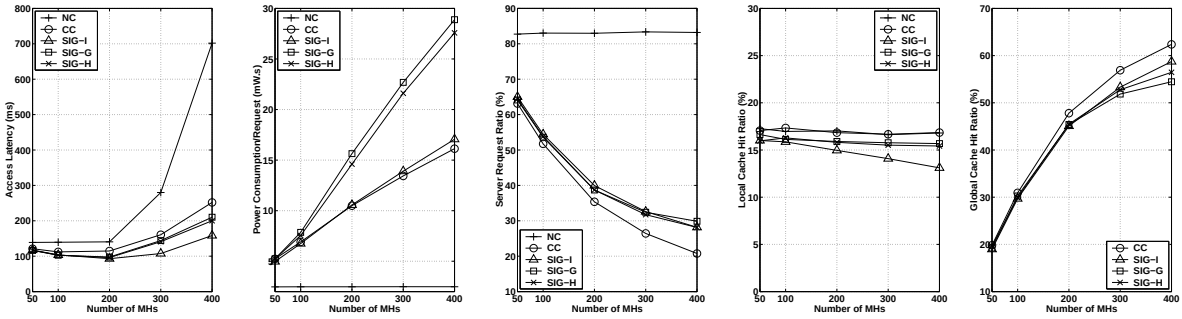
When the network traffic dynamically increases with increasing client movement speed, the timeout mechanism described in Section 3.3.2 may not be effective in detecting a GCH, due to occasional network congestion caused by exchanging cache signatures among MHs, when some MHs discover any link failure with their peers. This problem is referred to as a false negative error: no peer turns in the desired data item to the requesting MH upon a timeout, but in fact there are some peers caching it. To alleviate the false negative error, we can increase the network congestion factor, φ'.

In the environment with high client movement speed, there is a higher chance for an MH to encounter link failure with other peers, so that the maintenance overhead of SIG-G and SIG-H gets higher with increasing client movement speed. Thus, the

MHs adopting SIG-I consume less power than the other CC-SIG schemes, as SIG-I is more effective in maintaining the stored cache signature than the others.

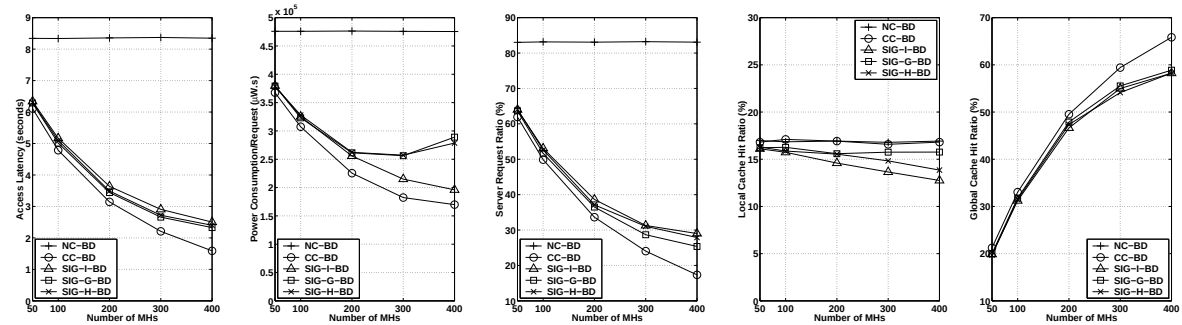
In the push-based and hybrid environments, we can draw similar conclusion as in the pull-based environment. The result suggests that SIG-I is more well-adapted to the various movement speed compared with the other CC-SIG schemes.

4.8.8 Effect of Number of MHs



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

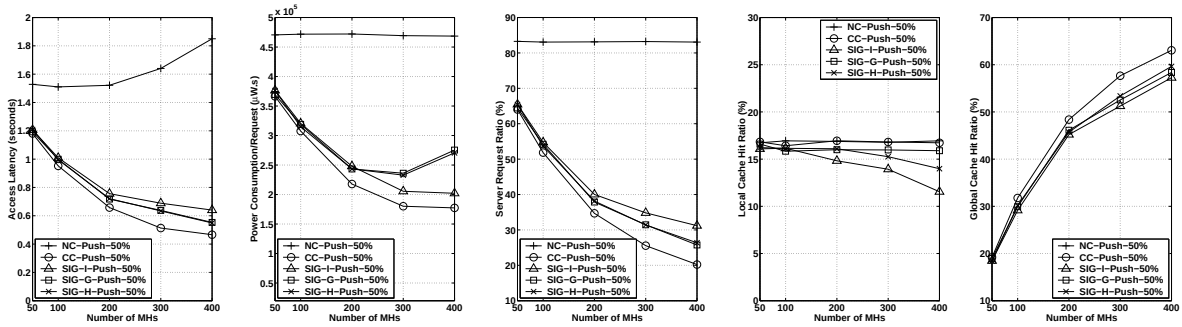
Figure 4.24: Effect of number of MHs in a pure pull-based environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.25: Effect of number of MHs in a pure push-based environment.

In this experiment, we study the effect of client population on system performance by increasing the number of MHs in the system from 50 to 400.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 4.26: Effect of number of MHs in a hybrid environment.

In the pull-based environment, the system workload becomes higher with increasing number of MHs in the system, so that access latency increases in all schemes, as depicted in Figure 4.24(a). In terms of access latency and server request ratio, the COCA schemes perform much better than NC, as the number of MHs gets larger. This is because there is a higher chance for the MHs to obtain the required data items from their peers, i.e., a higher GCH ratio, with increasing number of MHs. Thus, the MSS in a system adopting COCA schemes handles fewer requests, as shown in Figure 4.24(c). In other words, the system workload caused by increased number of MHs is shared among MHs. Therefore, we suggest that COCA can be used as an indirect load sharing technique in mobile environments. However, the drawback of the COCA schemes is that they are power-intensive protocol. As depicted in Figure 4.24(b), the power consumption of the MHs adopting COCA schemes increases with increasing number of MHs. The MHs have to consume much more power to turn in more required data items to the requesting peers, to receive more broadcast requests from peers and to discard a larger amount of unintended messages. The access latency and power consumption of SIG-G and SIG-H gets worse than SIG-I in heavily-loaded environments because there is a higher maintenance overhead for the MHs adopting SIG-G and SIG-H. The maintenance overhead leads to more message

passing among MHs and higher network traffic, so the MHs suffer from longer access latency and higher power consumption.

Since a push-based data dissemination model is scalable, the system performance is not degraded by increasing number of MHs, as shown in Figure 4.25(a). The performance of CC and SIG-I gets better with increasing number of MHs in the system. When the client population increases, there is a higher chance for the MHs to obtain the their desired data items from the peers, thereby achieving a higher GCH ratio, as depicted in Figure 4.25(e). The MHs adopting CC and SIG-I conserve more power with increasing GCH ratio, due to fact that the power consumption of a global cache access is much lower than a broadcast channel access. However, Figure 4.25(b) exhibits that the power consumption of SIG-G and SIG-H initially reduces with increasing client population, but it increases as the client population further gets larger. The initial drops arise because the MHs can take advantage of a higher GCH ratio. However, the rising power consumption as the number of MHs further increases, due to the maintenance overhead of SIG-G and SIG-H. When the number of MHs increases, there is a higher chance for the MHs detecting link failure with other peers. For SIG-G and SIG-H, when the MHs discover any link failure with peers whose cache signatures are stored in a counter vector, they have to reset the counter vector and re-collect the required cache signatures from the relevant peers to re-construct the counter vector, thus leading to higher power consumption and network traffic than CC and SIG-I.

In the hybrid environment, the behavior of the COCA schemes is similar as that in the push-based environment. However, when the number of MHs is over 200, the access latency of NC increases with increasing client population, as depicted in Figure 4.26(a). This is because the uplink channel is overloaded, leading to a longer channel waiting time. On the contrary, the access latency of the COCA schemes do not get worse with increasing number of MHs. This is because the COCA schemes

effectively reduce the number of requests sent to the MSS via the uplink channel, as many client requests can be handled by the MHs themselves.

4.9 Concluding Remarks

In this chapter, we propose a cache signature scheme for COCA. Cache signatures summarize the local cache content of the MHs. After exchanging cache signatures, the signatures provide hints for the MHs to determine whether the required data items are likely held by their peers. We have described the signature structure, i.e., bloom filter, generation and compression mechanism and exchange protocol among MHs. Besides compressing all cache signatures before transmission to reduce the transmission cost of exchanging cache signatures between peers, the technique of incremental cache signature is adopted. An MH only sends its full cache signature to its newly admitted neighbors, and then it piggybacks the *signature update information* on the request that is broadcast to its all neighboring peers. When an MH receives a request with signature update information, it extracts the signature update information from the request and updates the peer's cached signature accordingly. As client disconnection is a general behavior in mobile environments, we enhance the cache signature exchange protocol to handle client disconnection by using time-stamping mechanism to validate the stored cache signatures.

The performance of COCA with cache signature scheme is extensively evaluated through a number of simulated experiments. The result shows that the cache signature scheme improves access latency in comparison with the traditional caching scheme and standard COCA in the pull-based environment. However, as a tradeoff, it increases the server request ratio because certain portion of cache space is used to store the peers' cache signatures, thereby reducing the effective cache size. The

result also indicates that the cache signature is not suitable to the push-based and hybrid environments, as the effective cache size is a key factor to system performance in these environments.

As the cache signature scheme induces communication and storage overheads, it is worthy to extend the usage of cache signatures to amortize the cost. So far we just adopt cache signatures to filter searching in the global cache. In the next chapter, we use cache signatures to perform cooperative cache replacement among a group of MHs, in order to improve effective cache size among a group of MHs. We will also propose an adaptive approach to reduce the storage overheads incurred by cache signatures.

Chapter 5

Group-based Cooperative Caching

5.1 Introduction

In mobile environments, client mobility and data access patterns are key factors to system performance and cache management strategies. The major issues in cooperative cache management are the proper choice of cache replacement and admission control strategies, so as to increase the data accessibility, not only with respect to individual client, but also to other peers. For instance, if an MH arbitrarily forwards its cached data items to its peers for cache replacement, the MH may not effectively obtain the forwarded data items, as the peers could have moved far away. Likewise, if an MH does not cache the data item returned by peers in order to conserve its cache, based on the belief that the peer would still be accessible in the future, it may have regretted as the peer moves far away. In addition, when an MH forwards a data item to another peer and they possess totally different data access patterns, the action will reduce the peer's LCH ratio, and the "alien" data item will be removed very soon, as the peer is not interested in that data item. The decision of whom a cached data item should be forwarded to thus depends on both factors of access affinity on data

items and the mobility pattern.

In the past, several distributed clustering algorithms have been proposed for mobile environments. The two simplest distributed clustering algorithms are the lowest-ID [29] and largest-connectivity (degree) [67] algorithms. In the lowest-ID clustering algorithm, the MHs form clusters with their neighboring peers by broadcasting their ID to them, and the MH with the lowest ID in a cluster is assigned as the clusterhead. In the largest-connectivity clustering algorithm, the MHs also form clusters with their neighboring peers, but the MH with the highest connectivity becomes the clusterhead. To break a tie, the lowest-ID clustering algorithm is performed.

The cluster information, i.e., clusterhead and cluster members, may change frequently, due to the fact that the network topology of MANETs dynamically changes. Gerla et al. [35] find that the lowest-ID algorithm is more stable than the largest-connectivity one. When the MHs roam freely, their connectivity change so that they need to re-select a new clusterhead. The authors in [35] adopt the lowest-ID algorithm to group MHs into clusters. Each clusterhead is dedicated to perform channel scheduling and power control for its own cluster, and establish virtual circuit connection for cluster-wise multimedia information access.

To improve the stability of clusters in MANETs, several mobility-based clustering algorithms are proposed. In [7], the authors propose a bottom-up cluster algorithm considering the mobility pattern of MHs. The MHs record their own mobility pattern in a mobility profile, and they periodically exchange their mobility profiles with other peers. The clustering criterion is based on the relative velocity between MHs. An MH with the lowest-ID and its average relative mobility between all neighboring peers being larger or equal to a predefined threshold is selected as a clusterhead. Then, the inter-cluster merging algorithm is executed to combine clusters that are within a distance of hops. In the simulated experiments, both the random and group-based

mobility patterns are considered. The result shows that the proposed mobility-based cluster algorithm is more stable than the lowest-ID and highest-connectivity clustering algorithms.

In [85], an incremental clustering algorithm is proposed to discover group mobility pattern, in order to alleviate network partitioning problem. The algorithm only considers user mobility pattern, and identifies all mobility groups in the system. When the system detects that some network partitions are likely to form, i.e., there is no MH acting as a gateway between two mobility groups, it replicates appropriate data items among mobility groups to improve system performance.

Another mobility-based clustering algorithm, namely DRAM [48] is proposed to improve data accessibility in the system. DRAM considers not only the current motion information of the MHs, but also their historical motion locations. When a cluster is constructed, the data replica allocation algorithm is executed to place appropriate data items in the cluster members to improve data accessibility.

Lam et al. propose another distributed mobility-based clustering algorithm, namely GBL [52], for group-based location update in mobile environments. In GBL, the MH with the highest degree of affinity in neighborhood is assigned as the cluster leader. The degree of affinity is defined by the distance between an MH and its peers, together with the similarity in their movement vectors. Each cluster leader is responsible for updating the locations of cluster members. To improve the cluster stability and reduce network traffic, location prediction technique is adopted to cluster admission control and intra-cluster location update (from cluster members to the leader) respectively. The cluster admission protocol of GBL is improved, in [54], to reduce the amount of message passing in the protocol. Additionally, to further improve the cluster stability, Lam et al. [53] extend the distributed clustering algorithm to select a stand-by cluster leader that will take over the responsibility of the primary leader

if necessary. The selection of a stand-by cluster leader considers both the mobility pattern and connectivity.

Other than using velocity or distance to determine the mobility pattern, the author in [13] proposes to adopt signal power detection to calculate the relative mobility metric for distributed clustering in MANETs. The calculation of the mobility metric is based on the received power at a receiving MH between two successive packet transmission from the same peer. The MH with the lowest aggregate local mobility value that is defined by the variance of the mobility metric with all the neighboring peers in the neighborhood becomes the cluster head.

In this chapter, we propose two group-based COCA schemes, one is a centralized scheme (called **CGCoca**), and the other one is distributed (called **DGCoca**). Both schemes define and make use of the concept of a *tightly-coupled group* (TCG) that is defined as a group of MHs that are geographically and operationally close, i.e., sharing common mobility and data access patterns. It is not difficult to define whether two MHs are geographically close, based upon their locations. Two MHs are said to be operationally close, if they perform similar operations and access similar set of data items. Since we are more interested in data management and caching issues in this work, we consider two MHs to be operationally close based on the set of data items they access. Also, in the group-based COCA schemes, two cooperative cache management protocols: *cooperative cache admission control* and *cooperative cache replacement*, are proposed for the MHs to work together to manage their cache space as an aggregate cache or global cache in a TCG.

5.2 CGCoca

In this section, we present a centralized group-based COCA scheme (CGCoca) [23]. In CGCoca, the common mobility pattern is discovered with an incremental clustering algorithm and the similarity in access pattern is captured by a vector space model [9].

5.2.1 Similarity Measurement in Mobility Patterns

The MSS performs the incremental clustering algorithm to cluster the MHs into TCGs based on their mobility patterns. The mobility pattern is modelled by the weighted average distance between any two MHs. The MHs need not explicitly update their locations, but they piggyback the location information on the request sent to the MSS when they are requesting data items from it. The location information is represented by a coordinate (x, y) that can be obtained by a global positioning system (GPS) [36] or indoor sensor-based positioning systems, such as BAT [44] and Cricket [68]. For two MHs, m_i and m_j, the distance between them is calculated as the Euclidean distance, $|m_i m_j| = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$, where (x_i, y_i) and (x_j, y_j) are the coordinates of m_i and m_j respectively. An exponentially weighted moving average (EWMA) [21] is used to forecast the future distance of each pair of MHs based on their mobility histories. The weighted average distance between two MHs, m_i and m_j, is denoted by $||m_i m_j||$. It is initially set to $+\infty$. After the MSS receives both the location information of m_i and m_j, $||m_i m_j||$ is set to $|m_i m_j|$. Then $||m_i m_j||$ is updated when either m_i or m_j sends its new location to the MSS, based on the equation:

$$\|m_i m_j\|^{new} = \omega \times |m_i m_j| + (1 - \omega) \times \|m_i m_j\|^{old}, \tag{5.1}$$

where ω $(0 \leq \omega \leq 1)$ is a parameter to weight the importance of the most recent distance. The weighted average distance of each pair of MHs is stored in a two-

dimensional matrix, called a *distance matrix* (DM).

5.2.2 Similarity Measurement in Data Access Patterns

Other than the mobility pattern of the MHs, the incremental clustering algorithm also considers the similarity of their accesses to the data items. In the MSS, each data item is associated with an identifier from 1 to $NumData$, where $NumData$ is the total number of data items stored in the server. The MSS maintains a counter vector (V) with a length $NumData$ for each MH.

When an MH accesses a data item, the MSS increments the corresponding counter for the MH. The similarity score of the access pattern of two MHs, m_i and m_j, is calculated by the equation:

$$\mathsf{sim}(m_i, m_j) = \frac{\sum_{d=1}^{NumData} V_i(d) \times V_j(d)}{\sqrt{\sum_{d=1}^{NumData} V_i(d)^2} \times \sqrt{\sum_{d=1}^{NumData} V_j(d)^2}}, \tag{5.2}$$

where $V_i(d)$ is the total number of times that an MH, m_i, accesses the data item d, and $0 \leq \mathsf{sim}(m_i, m_j) \leq 1$. The access pattern of two MHs are more similar to each other when the similarity score gets higher. If two MHs possess the same access pattern, the similarity score is equal to one. The similarity score of each pair of MHs is stored in a two-dimensional matrix, called an *access similarity matrix* (ASM).

Since the uplink channel is scarce, a passive approach is adopted for collecting the data access pattern. An MH does not send any data access information actively to the MSS, but the MSS learns its data access pattern from the received requests along with location information sent by the MH. If the measured similarity score of two MHs is larger than or equal to a threshold δ_c, they are considered to possess a similar access pattern. The threshold δ_c is set to the average non-zero similarity score in ASM, and adjusted with the standard deviation to adapt to client access behavior in different

types of systems, i.e., $\overline{s} = \frac{\sum_{i=1}^{NumData-1}\sum_{j=i+1}^{NumData}\mathsf{sim}(m_i,m_j)}{\sum_{i=1}^{NumData-1}\sum_{j=i+1}^{NumData}\lceil\mathsf{sim}(m_i,m_j)\rceil}$, $\delta_c = \overline{s} + \phi\sigma_s$, where ϕ is a system parameter. Since the calculation of δ_c is costly, δ_c is only updated periodically (see Algorithm 13). As the measured similarity score only takes into consideration a sample of the access pattern of each MH, it is smaller than the actual similarity score. The MSS would not start the clustering algorithm until δ_c grows up to non-zero.

5.2.3 Incremental Clustering Algorithm

The incremental clustering algorithm clusters the MHs into TCGs by considering their mobility and data access patterns. The MHs in the same TCG possess a tight relationship because they share the common mobility and data access patterns.

In the clustering algorithm, a distance threshold Δ_c is adopted to determine whether an MH should be assigned to one of the existing clusters or a new cluster should be created. Every member in a TCG possesses two properties: for any two MHs, m_i and m_j, $\|m_i m_j\| \leq \Delta_c$, and $\mathsf{sim}(m_i, m_j) \geq \delta_c$. The two matrices DM and ASM are used as inputs to the algorithm. The first MH classified into a new cluster is considered as the leader of a cluster, called *cluster leader*. Each cluster has only one leader, and the cluster leader may be changed when a new MH is classified into the cluster.

The clustering algorithm considers the MHs one at a time and either assigns them to the existing clusters or creates new clusters for them. Let there be $NumClient$ MHs in the system, $\mathcal{M} = \{m_1, m_2, \ldots, m_{NumClient}\}$, in $|\mathcal{C}|$ existing clusters, $\mathcal{C} = \{C_1, C_2, \ldots, C_{|\mathcal{C}|}\}$ with $|\mathcal{C}|$ corresponding cluster leaders, $\mathcal{L} = \{l_1, l_2, \ldots, l_{|\mathcal{C}|}\}$. Initially, a new cluster C_1 is created for the first MH, m_1, and m_1 becomes the cluster leader of C_1. Then, the other MHs are considered one by one for admission. For each newly admitted MH, m, the algorithm finds out the closest cluster leader by looking up the weighted average distance in DM. m is assigned to the closest cluster

on condition that the two properties hold, i.e., the weighted average distance between m and the cluster leader is less than or equal to Δ_c and their similarity score is larger than or equal to δ_c; otherwise, the algorithm finds out the next closest cluster leader. This procedure is repeated until an appropriate cluster is found or no cluster satisfies the conditions. If there is no suitable cluster for m, m becomes the cluster leader of a new cluster.

When an MH is assigned to one of the existing clusters, the algorithm has to decide whether the newly admitted MH should be the leader of that cluster. The decision is based on their connectivity. The connectivity of an MH, m, as denoted by $\mathsf{Conn}(m)$, is defined as the number of peers whose weighted average distance between m and them is less than or equal to Δ_c. For instance, assume that m is assigned to an existing cluster C_h. If $\mathsf{Conn}(m) > \mathsf{Conn}(l_h)$, m takes over the role as the cluster leader from l_h; otherwise, no transition is required.

When a transition of cluster leader occurs, the algorithm checks whether any neighboring clusters should be merged together. If an MH, m, becomes the cluster leader of a cluster, C_h, the algorithm finds out the closest cluster leader, l_k. If the weighted average distances between m and all MHs, including the leader, in cluster C_k are less than or equal to Δ_c and their data access similarity scores are larger than or equal to δ_c, C_h and C_k are merged together. The cluster leader of the merged cluster is the one with the higher connectivity. If they are tied, the initiator l_h becomes the leader of the cluster. The algorithm repeats the checking to the next closest cluster, until the weighted average distance between the next closest cluster and m is larger than Δ_c.

The MSS postpones the announcement of any changes in the cluster, e.g., an MH is assigned to a new cluster or an MH joins/leaves, to any affected MH until the MH sends a request to it. The MSS then piggybacks the up-to-date member list to the

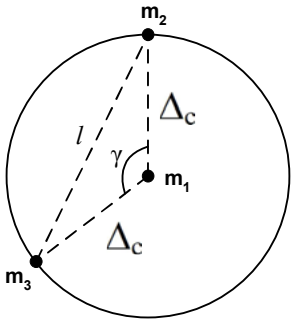


Figure 5.1: Distance threshold (Δ_c) selection

MH along with the required data item. In the member list, the first MH is the cluster leader. In effect, we are performing an asynchronous group view change [14], without enforcing stringent consistency among group members.

The incremental clustering algorithm consists of four components (Algorithms 13, 14, 15 and 16). Algorithm 13 is a continuous clustering algorithm that is periodically executed to keep track of any changes in the MH mobility and data access patterns and hence the TCG properties. Algorithm 14 is executed when the system detects a new MH. Algorithm 15 is invoked when an MH leaves the system. Algorithm 16, which is invoked by Algorithm 13, is dedicated to handling cluster merging operations.

In the incremental clustering algorithm, the distance threshold Δ_c is a key factor to clustering results. If we consider a case that all MHs in a cluster can connect to each other in single-hop communication, the required threshold distance Δ_c is now derived. Figure 5.1 illustrates a cluster wherein m_1 is the leader of the cluster, and m_2 and m_3 are two MHs classified into m_1's cluster. Let the weighted average distance between m_2 or m_3 and m_1 be equal to Δ_c. The distance l between m_1 and m_2 can be computed as:

$$l = \sqrt{\Delta_c^2 + \Delta_c^2 - 2\Delta_c^2 \cos\gamma} = \Delta_c\sqrt{2(1-\cos\gamma)}, \tag{5.3}$$

where $0 < \gamma \leq 180°$. When $\gamma = 180°$, $l = 2\Delta_c$, i.e., l is maximal. To ensure single-hop

Algorithm 13 The continuous clustering algorithm for CGCoca

```
1: procedure ClusterUpdate(Cluster 𝒞, Leader ℒ, MH ℳ)
2: while true do
3:   // 𝒞 is a set of clusters, 𝒞 = {C_1, C_2, ..., C_|𝒞|}
4:   // ℒ is a set of leaders for the clusters in 𝒞, ℒ = {l_1, l_2, ..., l_|𝒞|}
5:   // ℳ is a set of MHs, m, in the system
6:   for all m ∈ ℳ do
7:     cid ← ClusterID(m); // return the ID of the assigned Cluster of an MH
8:     classified ← false;
9:     if l_cid = m then
10:      ClusterMerge(𝒞, ℒ, m);
11:    else
12:      while min_{C_j ∈ 𝒞} ||m l_j|| ≤ Δ_c do
13:        if sim(m, l_j) ≥ δ_c then
14:          if m ∉ C_j then
15:            // reassign m to C_j
16:            C_cid ← C_cid − {m};
17:            C_j ← C_j ∪ {m};
18:          end if
19:          if Conn(m) > Conn(l_j) then
20:            // change leadership
21:            l_j ← m;
22:            ClusterMerge(𝒞, ℒ, m);
23:          end if
24:          classified ← true;
25:        else
26:          next C_j;
27:        end if
28:      end while
29:      if not classified then
30:        // form a new cluster for m
31:        k ← newClusterID(); // return a new ID
32:        C_cid ← C_cid − {m};
33:        C_k ← {m};
34:        𝒞 ← 𝒞 ∪ {C_k};
35:        l_k ← m;
36:        ℒ ← ℒ ∪ {l_k};
37:      end if
38:    end if
39:  end for
40: end while
```

Algorithm 14 The MH join algorithm for CGCoca

```
1: procedure MHJoin(Cluster 𝒞, Leader ℒ, MH m)
2: // 𝒞 is a set of clusters, 𝒞 = {C_1, C_2, ..., C_|𝒞|}
3: // ℒ is a set of leaders for the clusters in 𝒞, ℒ = {l_1, l_2, ..., l_|𝒞|}
4: // m is a new MH in the system
5: classified ← false;
6: if 𝒞 ≠ φ then
7:    while min_{C_j ∈ 𝒞} ||m l_j|| ≤ Δ_c do
8:       if sim(m, l_j) ≥ δ_c then
9:          C_j ← C_j ∪ {m};
10:         if Conn(m) > Conn(l_j) then
11:            l_j ← m;
12:            ClusterMerge(𝒞, ℒ, m);
13:         end if
14:         classified ← true;
15:      else
16:         next C_j;
17:      end if
18:   end while
19: end if
20: if not classified then
21:   // form a new cluster for m
22:   k ← newClusterID(); // return a new ID
23:   C_k ← {m};
24:   𝒞 ← 𝒞 ∪ {C_k};
25:   l_k ← m;
26:   ℒ ← ℒ ∪ {l_k};
27: end if
```

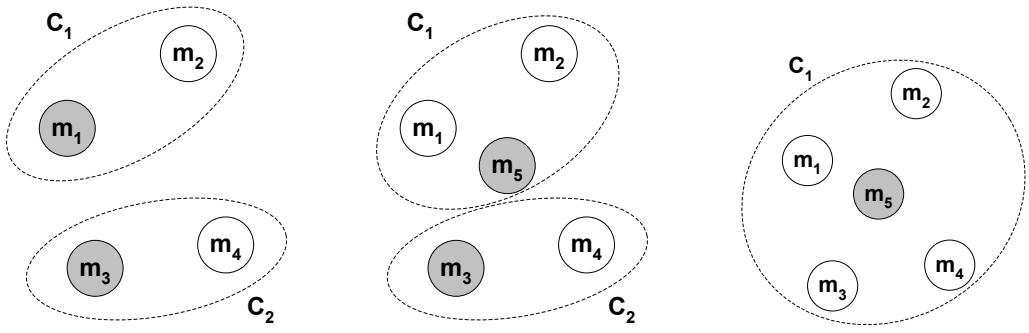
communication for the peers in a cluster, l should be less than or equal to $TranRange$, i.e., $2\Delta_c \leq TranRange$; hence, $\Delta_c = \frac{TranRange}{2}$. If Δ_c is set to $TranRange$, all MHs in a cluster can still connect to one another in single- or two-hop communication.

A simplified example for the clustering algorithm is depicted in Figure 5.2. There are two clusters and four MHs in the system. The grey nodes are the cluster leaders. Let the data access similarity scores among them be larger than or equal to δ_c so that we can merely consider their weighted average distances in this example. The average weighted distance between any two of them are: $\|m_1m_2\|, \|m_3m_4\| \leq \Delta_c <$

Algorithm 15 The MH leave algorithm for CGCoca

```
1: procedure MHLeave(Cluster C, Leader L, MH m)
2: // C is a set of clusters, C = {C_1, C_2, ..., C_|C|}
3: // L is a set of leaders for the clusters in C, L = {l_1, l_2, ..., l_|C|}
4: // m is a leaving MH in the system
5: cid ← ClusterID(m);
6: C_cid ← C_cid − {m};
7: if C_cid = φ then
8:    C ← C − {C_cid};
9:    L ← L − {l_cid};
10:   return;
11: end if
12: if l_cid = m then
13:   k ← arg_j max_{m_j ∈ C_cid}(Conn(m_j));
14:   l_cid ← m_k;
15: end if
```

$\|m_1m_3\|, \|m_1m_4\|, \|m_2m_3\|, \|m_2m_4\| \leq TranRange$. As a result, m_2 and m_4 are assigned to clusters C_1 and C_2 respectively. Consider a newly admitted MH, m_5, as shown in Figure 5.2(b). Let the weighted average distance between m_5 and any other MHs be also less than Δ_c. As $\|l_1m_5\|$ is shorter than $\|l_2m_5\|$, m_5 is assigned to cluster C_1. A transition of the cluster leader in C_1 needs to be considered. Since $\mathsf{Conn}(m_5)(=4)$ is larger than $\mathsf{Conn}(l_1)(=2)$, m_5 then becomes the new leader of C_1. After a new leader is elected, the clustering algorithm checks whether any other clusters should be merged to the cluster with a new leader. The clustering algorithm



(a) Two existing clusters **(b)** Clustering a new MH **(c)** Merging two clusters

Figure 5.2: An example of the incremental clustering algorithm.

Algorithm 16 The cluster merge algorithm for CGCoca

```
1: procedure ClusterMerge(Cluster C, Leader L, MH m)
2: // C is a set of clusters, C = {C_1, C_2, ..., C_|C|}
3: // L is a set of leaders for the clusters in C, L = {l_1, l_2, ..., l_|C|}
4: // m is a cluster leader that initiates the cluster merge algorithm
5: cid ← ClusterID(m);
6: while min_{C_j ∈ C ∧ cid ≠ j} ||l_cid l_j|| ≤ Δ_c do
7:    if sim(l_cid, l_j) ≥ δ_c then
8:       merge ← true;
9:       for all m_h ∈ C_j do
10:         if ||l_cid m_h|| > Δ_c or sim(l_cid, m_h) < δ_c then
11:            merge ← false;
12:            break;
13:         end if
14:      end for
15:      if merge then
16:         if Conn(l_cid) < Conn(l_j) then
17:            l_cid ← l_j;
18:         end if
19:         C_cid ← C_cid ∪ {C_j};
20:         C ← C − {C_j};
21:         L ← L − {l_j};
22:      end if
23:   end if
24: end while
```

first checks the closest cluster C_2 because the average weighted distance between l_1 and l_2 is the shortest and less than Δ_c. The distance between l_1 and all other MHs in C_2, i.e., m_4, is less than Δ_c, so C_2 is merged into C_1. As m_5 has the highest connectivity in the merged cluster, it remains as the leader of the cluster, as depicted in Figure 5.2(c). After presenting the clustering algorithm, CGCoca with cache signature scheme will be described in the next section.

5.2.4 CGCoca with Cache Signature Scheme

This scheme is extended from the cache signature scheme to save storage overhead by considering the group-based mobility pattern. In CGCoca, the MHs adopt counter

vector structure with dynamic counter size to store cache signatures sent from their peers. The MHs only exchange their signatures with other peers who are belonging to their own cluster to enhance the stability of the counter vector. Each MH maintains a vector of M counters, i.e., M is the size of a bloom filter, each counter with $counter\_size$ bits. If an MH does not have any peers in its assigned cluster, $counter\_size$ is zero, i.e., the MH needs not to store any peer signature. Once there is a peer in the MH's cluster, it creates a counter vector with one bit and set all counters to zero. If the MH discovers a newly joined member in its cluster, it sends a $SigRequest$ message to it. The peer receiving $SigRequest$ returns its full cache signature to the requesting MH. Then, the MH updates the counter vector by incrementing the counters at the position of the bits that are set in the received cache signature. If there are more than one peers assigned to the MH's cluster at the same time, the MH requests their cache signatures and updates the counter vector one by one. When the MH is going to increase a counter to a value of $2^{counter\_size}$, $counter\_size$ will be increased by one to avoid an overflow error. Likewise, in case that all counter values fall below $2^{counter\_size-1}$, $counter\_size$ will be decremented to save on cache space.

Algorithm 17 shows the mechanism of handling cluster member admission. Algorithm 18 describes the mechanism on how an MH handles a member leaving its cluster. After the MH synchronizes the cluster membership information with the MSS, when it detects a member leaving its cluster, it resets the counter vector, and then re-collects all cache signatures from its cluster members. The MH broadcasts $SigRequest$ to its neighborhood with the membership information. The peers receive the request and look up their identities in the membership information. If a peer finds that its identity is included in the membership information, it sends its full cache signature to the MHs. Otherwise, the peer drops the request.

Algorithm 17 Cache Signature Exchange Protocol for CGCoca - member admission at MH m_i

```
1: procedure OnDetectNewMember(CounterVector V_i, Peer P_i, Cluster C_i, MH m_j)
2: // V_i = {v_1, v_2, ..., v_M}
3: // P_i is a set of identifiers of MHs whose cache signatures are stored in m_i's cache
4: // C_i is a set of identifiers of MHs that are assigned to m_i's cluster
5: // m_j is a newly admitted cluster member in C_i
6: // Sig_j is the cache signature of m_j, Sig_j = {s_1, s_2, ..., s_M}
7: if m_j ∈ C_i then
8:    Send SigRequest to m_j;
9:    Receive Sig_j from m_j;
10:   P_i ← P_i ∪ {m_j};
11:   for all s_h ∈ Sig_j do
12:      if s_h = 1 then
13:         if v_h = 2^counter_size_i − 1 then
14:            counter_size_i ← counter_size_i + 1;
15:         end if
16:         v_h ← v_h + 1;
17:      end if
18:   end for
19: end if
```

Similar to the cache signature scheme, the cluster-based cache signature scheme should be able to handle client disconnection in CGCoca. When an MH broadcasts a request with the signature update information to its peers, it records the time to a variable, *last_signature_update_ts*. In addition, when an MH detects that it is disconnected from the network, it immediately records the time to another variable *disconnection_ts*. After a disconnected MH re-connects to the network, it sends a request to the MSS to synchronize its cluster membership information. The MH broadcasts *SigRequest* with its *disconnection_ts* and membership information to its neighbors. The peers who are belonging to the same cluster of the MH reply the MH with an acknowledgement message, if $last\_signature\_update\_ts < disconnection\_ts$. Otherwise, the peers turn in their full cache signatures to the MH. If the MH receives some cache signatures, it resets the counter vector and sends *SigRequest* to every

Algorithm 18 Cache Signature Exchange Protocol for CGCoca - member leave at MH m_i

```
1: procedure OnDetectMemberLeave(CounterVector V_i, Peer P_i, Cluster C_i, MH m_k)
2: // V_i = {v_1, v_2, ..., v_M}
3: // P_i is a set of identifiers of MHs whose cache signatures are stored in m_i's cache
4: // C_i is a set of identifiers of MHs that are assigned to m_i's cluster
5: // m_k is a leaving MH from C_i
6: if m_k ∈ P_i then
7:    for all v_i ∈ V_i do
8:       v_i ← 0;
9:    end for
10:   P_i ← ∅;
11:   counter_size_i ← 0;
12:   // re-collect cache signatures from all remaining cluster members in C_i;
13:   for all m_j ∈ C_i do
14:      Send SigRequest to m_j;
15:      Receive Sig_j from m_j;
16:      P_i ← P_i ∪ {m_j};
17:      for all s_h ∈ Sig_j do
18:         if s_h = 1 then
19:            if v_h = 2^counter_size_i − 1 then
20:               counter_size_i ← counter_size_i + 1;
21:            end if
22:            v_h ← v_h + 1;
23:         end if
24:      end for
25:   end for
26: end if
```

peer that responded with an acknowledgement message. The peer receives the request returns its full cache signature to the MH. After that, the MH re-constructs the counter vector based on the received cache signatures.

5.3 DGCoca

In this section, we present a distributed group-based COCA scheme (DGCoca) [26] that is also extended from COCA. It is similar to CGCoca; a group of MHs that

is sharing common mobility and data access patterns forms a TCG to manage their cached data items cooperatively. DGCoca exhibits two desirable features over CG-Coca. First, it is fully distributed, i.e., it does not need any help of the MSS. Second, the MH does not depend on any location systems or devices, e.g., GPS or other sensor-based positioning systems, to capture the mobility pattern of itself or other MHs. Thus, DGCoca is suitable for most indoor or open environments. The MHs adopting DGCoca use a *stable neighbor discovery algorithm* (SND) to find out the MHs with common mobility pattern, and they figure out the data access similarity between themselves and their peers by exchanging signatures that store their data access histories.

5.3.1 Stable Neighbor Discovery Algorithm (SND)

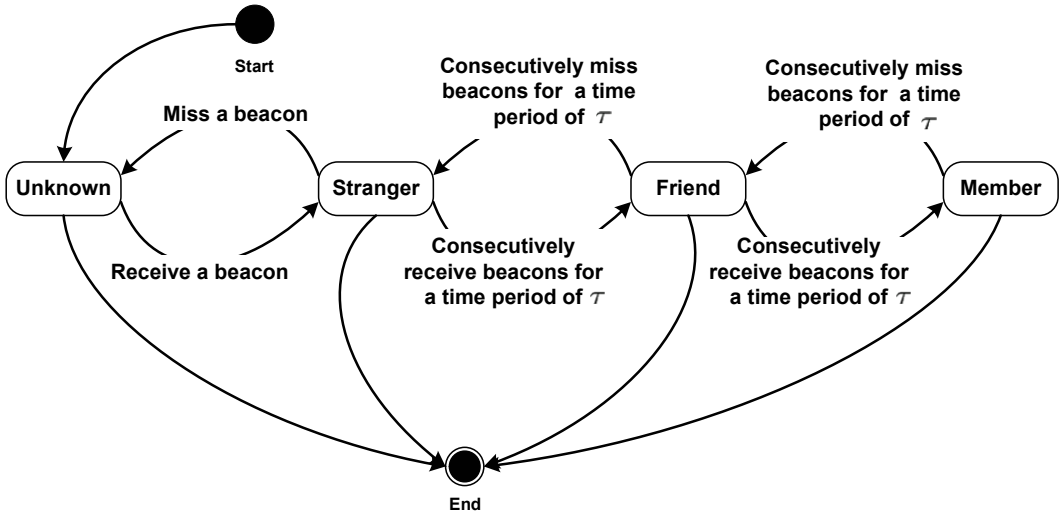


Figure 5.3: A state diagram of the client relationship

In DGCoca, we propose a memorization-based SND that is extended from NDP for discovering stable neighboring peers. In SND, there are three kinds of relationship between any two MHs: *stranger*, *friend* and *member* (in an increasing order of the

Algorithm 19 Stable Neighbor Discovery Algorithm for DGCoca at MH m_i

```
1: procedure DiscovryStableNeighbors(CounterVector V_i, Relation R_i, Peer P_i,
   Score S_i)
2: // V_i = {v_1, v_2, ..., v_M}
3: // P_i is a set of identifiers of MHs whose cache signatures are stored in m_i's cache
4: // rel(i, j) maintains the current relationship between m_i and m_j
5: // R_i contains only relationship with m_i, i.e., ∀k = i ∧ i ≠ j, rel(k, j) ∈ R_i
6: // S_i maintains the valid similarity scores of access pattern of m_i's peers
7: for all rel(i, j) ∈ R_i do
8:    // now() returns the current time
9:    if now() − last_beacon_ts_j > T_Beacon then
10:      if rel(i, j) = "stranger" then
11:         rel(i, j) ← "unknown";
12:         R_i ← R_i − {rel(i, j)};
13:         OnDetectTCGMemberLeave(V_i, P_i, m_j);
14:         next m_j;
15:      end if
16:      if not disappeared_j then
17:         last_disappeared_ts_j ← now();
18:         disappeared_j ← true;
19:      end if
20:   end if
21:   if disappeared_j then
22:      if now() − last_disappeared_ts_j ≥ τ then
23:         if rel(i, j) = "member" then
24:            rel(i, j) ← "friend";
25:         else if rel(i, j) = "friend" then
26:            rel(i, j) ← "stranger";
27:         end if
28:         last_disappeared_ts_j ← now();
29:      end if
30:   else
31:      if now() − first_consecutive_beacon_ts_j ≥ τ then
32:         if rel(i, j) = "stranger" then
33:            rel(i, j) ← "friend";
34:            first_consecutive_beacon_ts_j ← now();
35:         else if rel(i, j) = "friend" then
36:            if CheckSimiliarityScore(S, m_i, m_j) then
37:               // procedure CheckSimiliarityScore() will be described in Section 5.3.2
38:               rel(i, j) ← "member";
39:               OnDetectNewTCGMember(V_i, P_i, m_j);
40:            end if
41:         end if
42:      end if
43:   end if
44: end for
```

Algorithm 20 Stable Neighbor Discovery Algorithm for DGCoca at MH m_i

```
1: procedure OnReceiveBeacon(Relation R_i, MH m_j)
2: // rel(i,j) maintains the current relationship between m_i and m_j
3: // R_i contains only relationship with m_i, i.e., ∀k = i ∧ i ≠ j, rel(k,j) ∈ R_i
4: // m_i receives a "hello" beacon message from m_j
5: last_beacon_ts_j ← now();
6: if rel(i,j) ∉ R_i then
7:    rel(i,j) ← "stranger";
8:    R_i ← R_i ∪ {rel(i,j)};
9:    first_consecutive_beacon_ts_j ← now();
10: else if disappeared_j then
11:    disappeared_j ← false;
12:    first_consecutive_beacon_ts_j ← now();
13: end if
```

closeness of relationships between two MHs). When an MH has communicated with a peer, i.e., the MH has consecutively received "hello" beacon from the peer, for a period of time, the relationship between them becomes closer. On the other hand, if a peer has disappeared to the MH for a period of time, their relationship will be degraded.

For two MHs, m_i and m_j, m_j becomes a "stranger" to m_i, if m_i receives a "hello" beacon from m_j, and m_i does not have any relationship information about m_j at that moment. When m_i consecutively receives "hello" beacons from m_j for an additional period of time, τ, m_j becomes m_i's "friend". Then, if m_i obtains "hello" beacons from m_j consecutively for another period of time, τ, m_i treats m_j as its "member" in its TCG. Nevertheless, if m_i cannot receive any beacon from m_j beyond a time period of τ, their relationship is downgraded to "friend". After another period of time τ, m_i still cannot receive any beacon from m_j, the memory of m_i for m_j is faded away, and m_j becomes a "stranger" to m_i. m_i next removes the relationship information about m_j, and m_j reverts to be *unknown* to m_i. Figure 5.3 shows the state diagram of the client relationship in DGCoca.

SND consists of two components (Algorithms 19 and 20). Algorithm 19 is a con-

tinuous algorithm that is periodically executed to learn about any changes in the relationship between other MHs. Algorithm 20 is executed when the MH receives a "hello" beacon from its neighboring peers.

5.3.2 Similarity Measurement in Data Access Patterns

In DGCoca, an *access history signature* is used to store the access history of an MH. When the MH accesses a data item, it generates a data signature for it. Then, the access history signature is updated by performing a bitwise OR operation on the data signature with the last access history signature, i.e., (access history signature)^{new} = data signature $\oplus$ (access history signature)^{old}. The data access pattern introduces an additional condition to the SND algorithm. When an MH, says m_i, is going to upgrade a relationship with a peer, m_j from "friend" to "member", it requests the peer to turn in its access history signature. The MH calculates a score on the data access similarity between themselves by Jaccard Coefficient,

$$\mathsf{sim}(m_i, m_j) = \frac{A_i \cap A_j}{A_i \cup A_j},\ 0 \leq \mathsf{sim}(m_i, m_j) \leq 1, \tag{5.4}$$

where A_i and A_j are the access history signatures of m_i and m_j respectively. If $\mathsf{sim}(m_i, m_j)$ is equal to or larger than the similarity threshold δ_d, the peer becomes the MH's "member"; otherwise, the peer only remains to be its "friend". In DGCoca, a small value is set to δ_d because we would like to obtain a higher GCH ratio, and only filter out the peers whose access patterns are totally different.

In an access history signature, since k bits are randomly assigned to each data item, some bits may be used to represent a part of multiple data items. When two data items share some common bits in an access history signature, the similarity score of these data items is larger than zero, even though they are two totally different

Algorithm 21 Checking the similarity score between two MHs for DGCoca

```
1: procedure CheckSimiliarityScore(Score S, MH m_i, MH m_j)
2: // S is a set of valid similarity scores of access pattern of peers maintained by m_i
3: if score_j ∈ S then
4:    if now() − last_score_ts_j ≤ SimScoreTTL_j then
5:       if score_j ≥ δ_d then
6:          return true;
7:       else
8:          return false;
9:       end if
10:   else
11:      S ← S − {score_j};
12:   end if
13: end if
14: // get m_j's access history signature
15: last_score_ts_j ← now();
16: S ← S ∪ {score_j};
17: if sim(m_i, m_j) ≥ δ_d then
18:    return true;
19: else
20:    return false;
21: end if
```

data items. This means that the estimated similarity score may be higher than the actual score. In fact, the two measures are not identical in nature. Fortunately, if the false positive probability is low, the estimated similarity score is a highly accurate approximation to the actual score. We do need this approximation, since it is very costly, if not impossible, to compute and compare the similarity based on an actual access history in practice.

To verify the appropriateness of the access history signature approach as an efficient and practical approximation, it is compared with the bit vector approach, in which each data item is assigned to a unique integral identifier from one to N. When a bit vector with N bits is used, each data item can be perfectly assigned to a particular bit based on its unique identifier. Thus, the MHs adopting the bit vector approach can precisely determine a data access similarity score between themselves and their peers.

The appropriateness of the access history signature approach is evaluated through a simulated experiment.

In our experiment, when an MH accesses a data item, it sets the corresponding bit in the bit vector and updates the access history signature. The simulated experiment is run for two million time units in an environment of 100 MHs. For consistency and accuracy, the details of the simulated experiment setting follows those in Section 5.5. We calculate the similarity score of each pair of MHs (there are $\frac{100\times99}{2} = 4{,}950$ pairs) by using the access history signature and bit vector approaches every two hundred thousand time units, i.e., there are $\frac{2{,}000{,}000}{200{,}000} \times 4,950 = 49,500$ data points. The two scores, (bit_vector_score, $access\_history\_signature\_score$), are recorded and plotted as a scatter plot in Figure 5.4. The straight line shown in the figure is the reference line ($y = x$) that we are looking for, i.e., direct relationship between bit vector and access history signature approaches. The result depicts that the estimated similarity score possesses a very strong linear relationship with the actual score. The better the linear relationship between the estimated and actual similarity scores is, the higher the accuracy of the estimation is. Therefore, the result confirms that the estimated data access similarity score obtained by using our proposed access history signature approach is a highly precise approximation to the actual score. Based on the data, we obtain a correlation coefficient of $r = \frac{\sum_{i=1}^{n}(x_i-\overline{x})(y_i-\overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\overline{x})^2 \sum_{i=1}^{n}(y_i-\overline{y})^2}} = 0.9994$, where n is the total number of points, $\overline{x}$ and $\overline{y}$ are the average value of the actual and estimated similarity scores respectively; that indicates a strong linear relationship between the estimated and actual similarity scores. Also, the correlation of determination is $r^2 = 0.9988$, which says that approximately 99% of the variation in the values of the estimated similarity score is accounted for by a linear relationship with the actual similarity score. The access history signature similarity score is thus a very good approximation to the actual score.

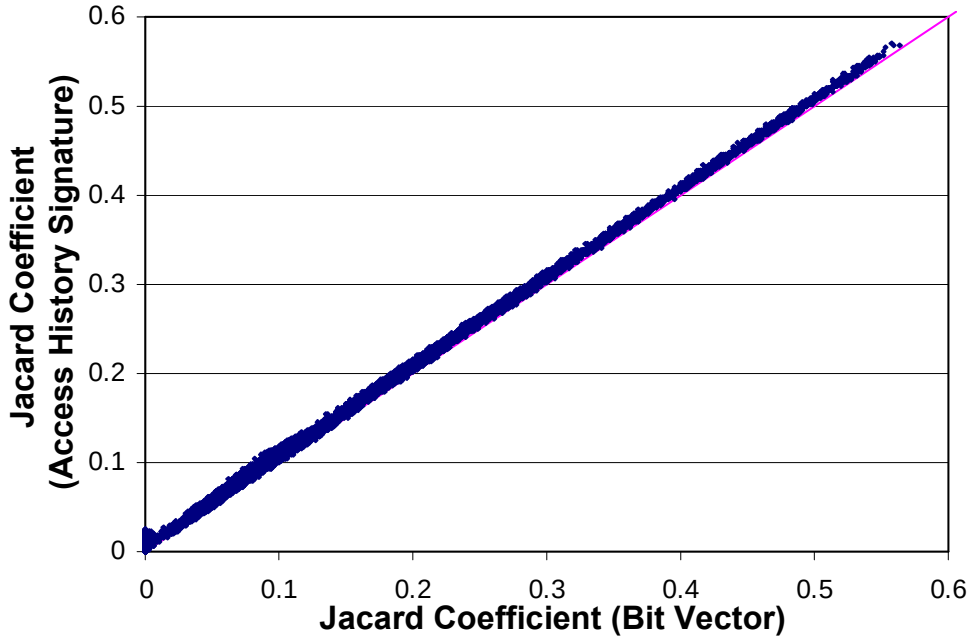


Figure 5.4: This figure depicts a scatter plot of 49,500 points of the estimated Jacard coefficient (access history signature approach, on the y-axis) and the actual Jacard coefficient (bit vector approach, on the x-axis).

To reduce communication overheads, a similarity score will be maintained for a period of time-to-live ($SimScoreTTL$). The calculated score is removed after its $SimScoreTTL$ elapses. Algorithm 21 checks whether the similarity score of two MHs is larger than δ_d. If so, it returns true to the calling algorithm (Algorithm 19); otherwise, it returns false.

5.3.3 DGCoca with Cache Signature Scheme

Since DGCoca is a distributed mechanism, the MHs detect member joins and leaves through SND protocol without any help from the MSS. Each MH maintains a counter vector structure with dynamic counter size to store cache signatures sent by their peers. The MHs only exchange their signatures with other peers who are belonging to their own TCG. Each MH maintains a vector of M counters, i.e., M is the size of a bloom filter, each counter with $counter\_size$ bits. If an MH does

Algorithm 22 Cache Signature Exchange Protocol for DGCoca - member admission at MH m_i

```
1: procedure OnDetectTCGNewMember(CounterVector V_i, Peer P_i, MH m_j)
2: // V_i = {v_1, v_2, ..., v_M}
3: // P_i is a set of identifiers of MHs whose cache signatures are stored in m_i's cache
4: // m_j is a newly admitted MH in m_i's TCG
5: // Sig_j is a cache signature from m_j, Sig_j = {s_1, s_2, ..., s_M}
6: if m_j ∉ P_i then
7:    Send SigRequest to m_j;
8:    Receive Sig_j from m_j;
9:    P_i ← P_i ∪ {m_j};
10:   for all s_h ∈ Sig_j do
11:      if s_h = 1 then
12:         if v_h = 2^counter_size_i − 1 then
13:            counter_size_i ← counter_size_i + 1;
14:         end if
15:         v_h ← v_h + 1;
16:      end if
17:   end for
18: end if
```

not have any members in its TCG, *counter_size* is zero, i.e., the MH needs not to store any peer signature. Once there is a member in the MH's TCG, it creates a counter vector with one bit and set all counters to zero. If the MH discovers any newly joined TCG member, it sends a *SigRequest* message to it. The peer receiving *SigRequest* returns its full cache signature to the requesting MH. Then, the MH updates the counter vector by incrementing the counters at the position of the bits that are set in the received cache signature. If there are more than one peer joining to the MH's TCG at the same time, the MH requests their cache signatures and updates the counter vector one by one. When the MH is going to increase a counter to a value of $2^{counter\_size}$, the *counter_size* will be increased by one to avoid an overflow error. Likewise, in case that all counter values fall below $2^{counter\_size-1}$, the counter size *counter_size* will be decremented to save on cache space. Algorithm 22 presents an algorithm for the MHs to handle a newly discovered TCG member.

Algorithm 23 Cache Signature Exchange Protocol for DGCoca - member leave at MH m_i

```
1: procedure OnDetectTCGMemberLeave(CounterVector V_i, Peer P_i, MH m_k)
2: // V_i = {v_1, v_2, ..., v_M}
3: // P_i is a set of identifiers of MHs whose cache signatures are stored in m_i's cache
4: // m_k is a leaving MH from m_i's TCG
5: if m_k ∈ P_i then
6:    for all v_i ∈ V_i do
7:       v_i ← 0;
8:    end for
9:    P_i ← P_i − {m_k};
10:   counter_size_i ← 0;
11:   // re-collect cache signatures from all remaining members in m_i's TCG;
12:   for all m_j ∈ P_i do
13:      Send SigRequest to m_j;
14:      Receive Sig_j from m_j;
15:      for all s_h ∈ Sig_j do
16:         if s_h = 1 then
17:            if v_h = 2^counter_size_i − 1 then
18:               counter_size_i ← counter_size_i + 1;
19:            end if
20:            v_h ← v_h + 1;
21:         end if
22:      end for
23:   end for
24: end if
```

Algorithm 23 is executed by the MH, when a peer departs from the MH's TCG. After the MH detects a departure of its TCG member, it resets the counter vector, and broadcasts *SigRequest* with its TCG membership information to its neighbors. If the peers receive the request and find that their identities are contained in the membership information, they return their full cache signatures to the requesting MH. Otherwise, they simply drop the request. After the MH collects all required cache signatures, it re-constructs the counter vector.

To deal with the client disconnection problem, the MHs adopting DGCoca have to backup all relationships between their peers when they are just disconnected from the network. After they re-connect to the network, they need to restore all the

relationships. They broadcast a relationship restore request message containing their peers' identities and the relationship to their peers. The peers who receive the request also restore the relationship, and then they send an acknowledgement message to the requesting MH. However, as disconnected peers cannot receive the request, they will lose the relationship, and they have to build up their relationship from scratch again.

In DGCoca, the MHs have to check for the validity of their cached cache signatures after they re-connect to the network. The checking mechanism is the same as in CGCoca. When an MH broadcasts a request with the signature update information to its peers, it records the time to a variable, $last\_signature\_update\_ts$. Additionally, when an MH disconnects from the network, it records the time to another variable, $disconnection\_ts$. After it re-connects to the network, $disconnection\_ts$ will be included in the broadcast request. The peers who are the MH's members reply the MH with an acknowledgment message, if $last\_signature\_update\_ts < disconnection\_ts$. Otherwise, the peers return their full cache signatures to the MH. If the MH receives some cache signatures, it resets its counter vector and broadcasts a $SigRequest$ message to every peer that responded with an acknowledgement message. The peer who receives the request returns its full cache signature to the MH. After collecting all required cache signatures, the MH re-constructs the counter vector. To reduce communication overhead, $SigRequest$ is combined with a relationship restore request message, i.e., the relationship restore request message contains the value of $disconnection\_ts$.

5.4 Cooperative Cache Management Protocols

We propose two cooperative cache management protocols: cooperative cache admission control and cooperative cache replacement, for the MHs to work together to manage their cache space as a whole, i.e., the *aggregate cache* or *global cache*.

5.4.1 Cooperative Cache Admission Control

When an MH encounters a local cache miss, it sends a request to its peers. If some peers turn in the required data item to the MH and the local cache is not full, the MH caches data item, no matter it is returned by a peer in the same TCG or not. However, if the local cache is full, the MH does not cache the data item when it is supplied by a peer in the same TCG, on the belief that data item can be readily available from the peer if needed. If the cache is full but the data item comes from a peer outside of its group, the MH would rather cache the data item in its local cache by removing the least valuable data item, since the providing peer may move far away in future. After a peer sends the required data item to the requesting MH, if they are belonging to the same TCG, the peer updates the last accessed time stamp of the data item, so that the item can have a longer TTL in the global cache. If there are more than one member caching the same required data item, the MH selects the member caching the data item with the longest TTL to update the last access time stamp of the data item. If more than one data item with the same TTL, the MH randomly selects one of them to update the last access time stamp.

5.4.2 Cooperative Cache Replacement

The proposed cooperative cache replacement protocol allows the MH to collaborate with its "members" to replace the least valuable data item with respect to the MH itself and other members in the same TCG. This scheme can satisfy the three desirable properties for caching [24]. First, the most valuable data items are always retained in the local cache. Second, in a local cache, a data item which has not been accessed for a long item, is replaced eventually. Third, in a global cache, a data item which "spawns" replica is first replaced to increase the effective cache size.

The LRU cache replacement policy is used. To retain valuable data items in

the local cache, only a number of *ReplaceCandidate* least valuable data items are selected as the candidates that are to be replaced. Among the candidates, the least valuable data item is first chosen, and the MH generates a data signature for that data item. The data signature is then compared with the peer signature by a bitwise AND operation. If the result is the same as the data signature, the data item is likely a replica in the global cache, so it is removed from the cache. Otherwise, the second least valuable data item is chosen, and so on and so forth. If no candidate is probably replicated in the TCG, the least valuable data item is removed from the cache.

There could be a wasting issue with the above arrangement, a data item without any replica could always be retained in the local cache, even though it will not be accessed again. If most data items cached in the local cache of an MH belong to such type of data item, the MH's cache will be populated with useless data items. As a result, the LCH ratio will be degraded. To solve this problem, a *SingletTTL* counter is associated with each candidate. The *SingletTTL* is initially set to *ReplaceDelay*. When the least valuable data item is not replaced because it does not have any replica, its counter is decreased by one. The cooperative cache replacement mechanism is skipped, if the counter value of the least valuable data item is equal to zero, and the MH simply drops that data item from the cache. The counter is reset to *ReplaceDelay*, when the data item is accessed by the MH itself or other members in its TCG.

5.5 Simulation Model

The system performance of CGCoca and DGCoca is evaluated through a series of simulated experiments. The simulation model is based on the model defined in the Chapters 3 and 4 with additional parameter settings for the group-based COCA

Table 5.1: Simulation parameters and default settings for the group-based COCA schemes.

Parameter	Description	Default Value
GroupSize	No. of MHs in a motion group	10
ϕ	System parameter used to determine the data access similarity threshold in CGCoca	0
Δ_c	Distance threshold in CGCoca	50 m
τ	Time period threshold in DGCoca	10 s
SimScoreTTL	Time-to-live of a similarity score in DGCoca	30 s
δ_d	System parameter used to determine the data access similarity threshold in DGCoca	0.05
ReplaceDelay	Replacement delay of the least valuable data item in the cache	2
ReplaceCandidate	Proportion of cached data items participating in the cooperative cache replacement protocol	20 data items

schemes, as shown in Table 5.1.

5.5.1 Mobility Model

In the simulated mobile environment, the MHs are divided into several motion groups, and each group consists of *GroupSize* MHs. The mobility of the MHs is based on a "reference point group mobility" (RPGM) model [18, 46]. RPGM is a random mobility model for a group of MHs and for an individual MH within its motion group. Each motion group has a logical center which represents a motion reference point for all MHs belonging to the group. The mobility of each motion group is defined according to "random waypoint" model [17, 18].

5.5.2 Data Access Pattern

We consider a group-based data access pattern among motion groups. In default simulation settings, the MHs belonging to the same motion group possess the same access range, and their individual access pattern to the data items is following Zipf distribution [92] with a skewness parameter. To evaluate the effect of group-based access pattern on system performance, we consider two additional types of group-based access patterns, as follows:

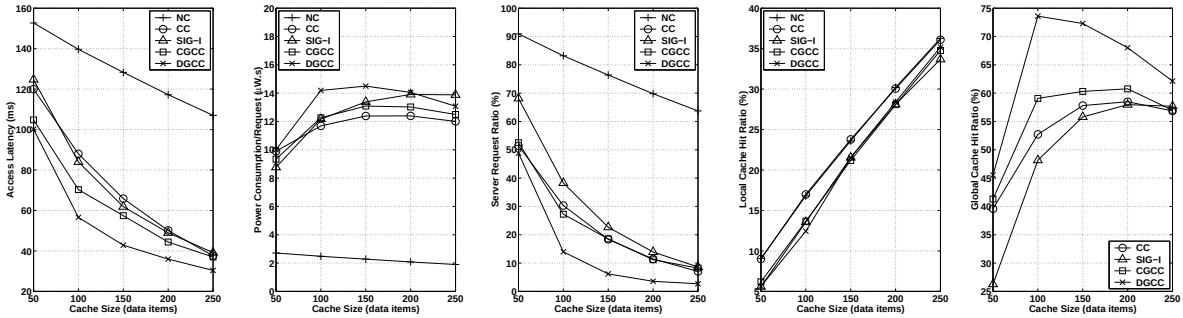
- Random access range - all MHs in a motion group have the same access range. The access range of each motion group is randomly selected.
- Common hot spot - all MHs in a motion group share a certain percentage of hot spot, i.e., common hot spot, and the remaining access ranges are randomly chosen for each MH. The common hot spot of each motion group is randomly selected.

5.6 Simulation Results

In the simulated experiments, we adopt the same simulation model, including power consumption, mobility, server and network models, as defined in Chapters 3 and 4 to compare the performance of the centralized and distributed group-based COCA with cooperative cache management protocols (denoted as CGCC and DGCC respectively) with a conventional caching scheme that does not involve any cooperation among MHs (denoted as NC), standard COCA (denoted as CC) and COCA with cache signature scheme that applies the individual signature storage scheme (denoted as SIG-I) in the pull-based, push-based and hybrid environments. All schemes adopt LRU cache replacement policy. For the push-based environment, only the broadcast

disk [2] broadcast scheduling algorithm is considered. Since CGCC relies on the MSS to discover TCGs, it cannot be applied to the push-based environment in which there is no uplink channel from the MHs to the MSS. Thus, we only examine the performance of CGCC in the pull-based and hybrid environments. All simulation results are recorded after the system reaches a stable state, in which all client caches are full, in order to avoid a transient effect. A simulated experiment terminates after each MH generates over 2000 requests after the warmup period. We conduct the series of experiments by varying several parameters: cache size, data item size, access patterns, client disconnection probability, mobility speed, number of MHs and motion group size. The performance metrics include access latency, power consumption, server request ratio, LCH ratio and GCH ratio.

5.6.1 Effect of Cache Size

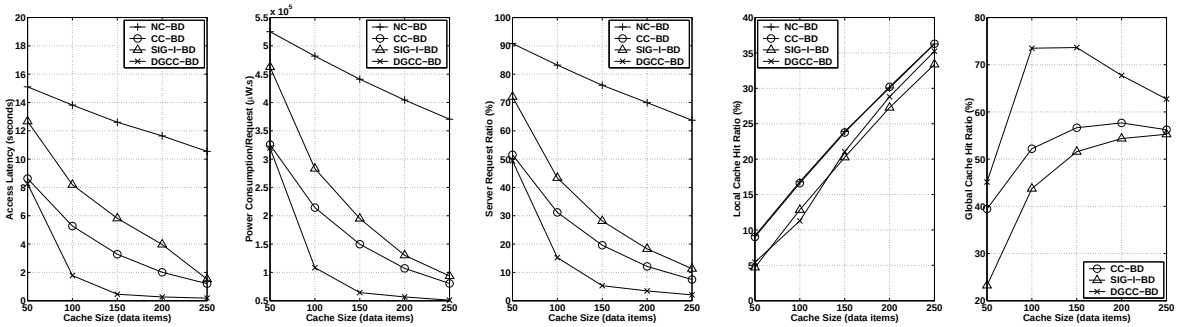


(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.5: Effect of cache size in a pure pull-based environment.

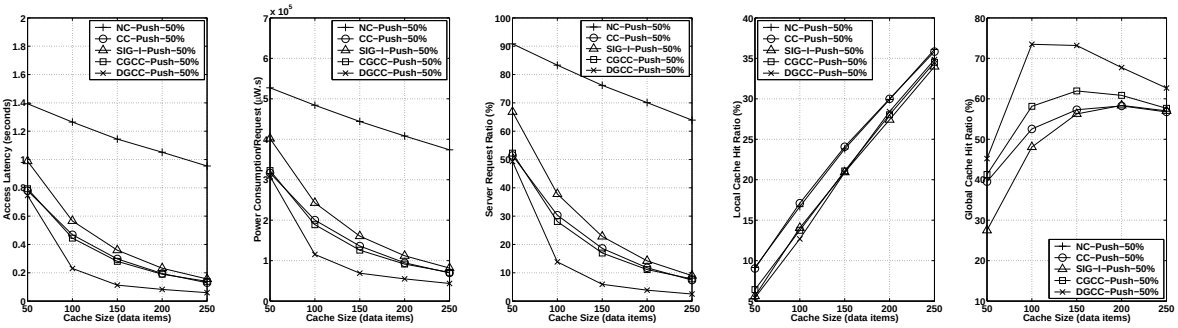
Our first simulated experiment studies the effect of cache size on system performance by varying the cache size from 50 to 250 data items.

Figures 5.5(a) and 5.5(c) show that the performance improves with increasing cache size. The LCH ratio increases as the cache size gets larger, as shown in Figure 5.5(d). The group-based COCA schemes perform better than the standard COCA in terms



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.6: Effect of cache size in a pure push-based environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.7: Effect of cache size in a hybrid environment.

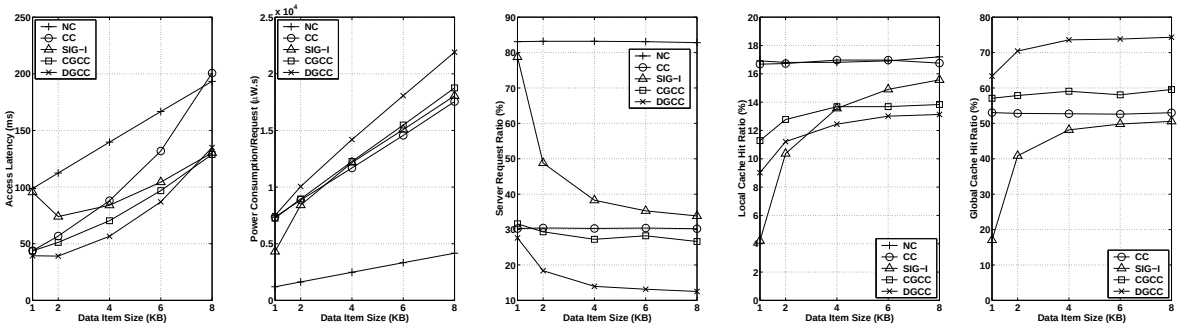
of access latency and server request ratio. In addition, DGCC performs better than CGCC, as its distributed membership formation approach gives faster response to the newly admitted peers than centralized one.

The cooperative cache management protocols on the aggregate cache reduce the LCH ratio of DGCC and CGCC, as shown in Figure 5.5(d), but it improves the data accessibility in TCGs, so that the MHs adopting group-based COCA schemes can enjoy a higher GCH ratio, as shown in Figure 5.5(e). In the pull-based environment, the cost of DGCC and CGCC is higher power consumption. When the MHs adopting these schemes enjoy a higher GCH ratio, they have to consume more power to return required data items to the requesting peers. Also, DGCC is more effective in discovering tightly-coupled peers than CGCC, so that for the DGCC MHs, they achieve a

higher GCH ratio, but they need to consume more power in handling global cache queries.

In the push-based and hybrid environments, DGCC outperforms the other schemes, in terms of access latency, power consumption and server request ratio. The effective cache size is a key factor to system performance in the push-based and hybrid environments. The group-based COCA schemes allow the TCG members to construct an aggregate cache, and manage it cooperatively by adopting the cooperative cache management protocols to ameliorate their effective cache size. As DGCC and CGCC improve the data accessibility of the aggregate cache, the MHs adopting them can achieve a higher GCH ratio than other schemes, as depicted in Figures 5.6(e) and 5.7(e). They record the best performance because the access latency and power consumption of a global cache access is much less than a broadcast channel access in the push-based and hybrid environments. Since DGCC is more effective in discovering TCGs than CGCC in a distributed environment, the performance of the MHs adopting DGCC is better than CGCC, as exhibited in Figures 5.6 and 5.7.

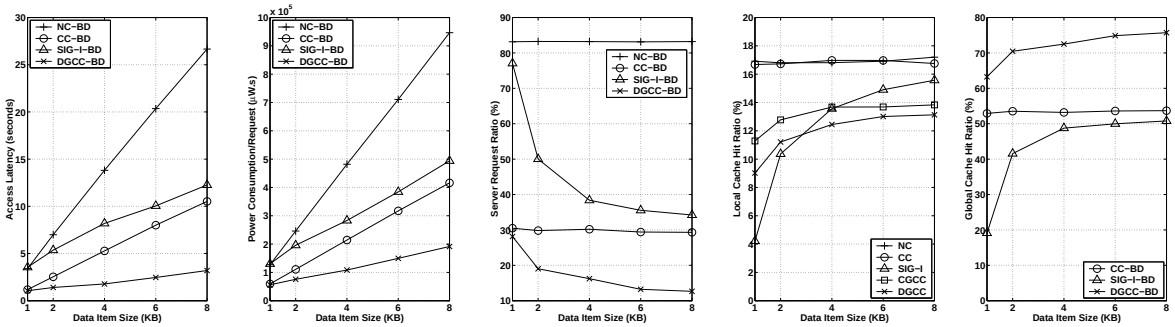
5.6.2 Effect of Data Item Size



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

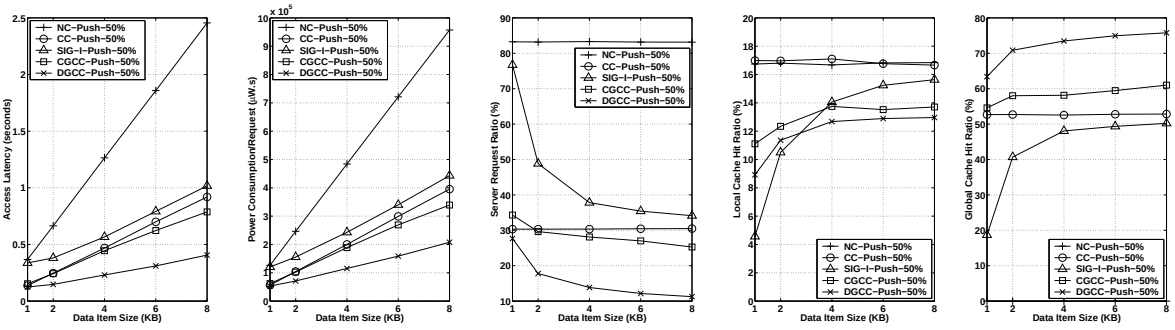
Figure 5.8: Effect of data item size in a pure pull-based environment.

We next study the effect of data item size on system performance by increasing



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.9: Effect of data item size in a pure push-based environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.10: Effect of data item size in a hybrid environment.

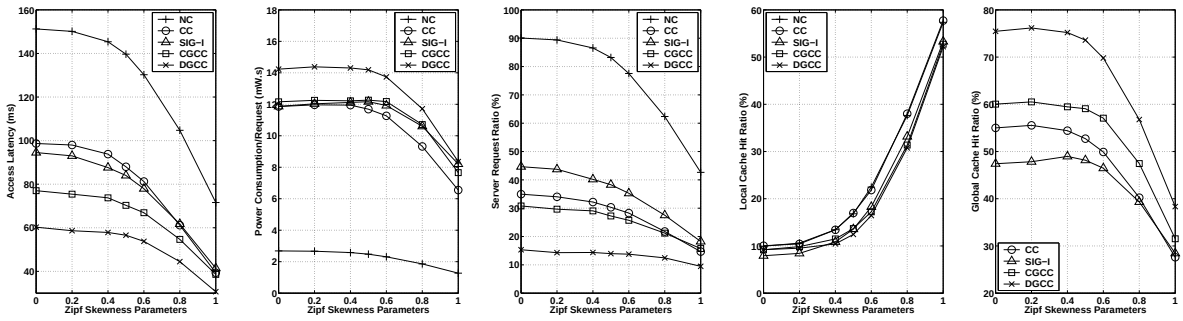
the data item size from one to eight kilobytes (KB).

Figures 5.8(a), 5.9(a) and 5.10(a) show that the access latency of all schemes gets worse with increasing data item size because the transmission time increases, as the data item size gets larger. Likewise, when the MHs transmit data items with larger size, they have to spend more power on the transmission, as shown in Figures 5.8(b), 5.9(b) and 5.10(b).

When the data item size gets larger, the ratio of cache space occupied by the cache signature to the total cache space decreases. SIG-I can perform better than CC, as the data item size is larger than 4 KB. In terms of access latency and server request ratio, DGCC and CGCC perform better than CC and SIG-I, and DGCC is the best scheme. It shows that the group-based COCA schemes effectively improve system

performance with various data item sizes. In group-based COCA schemes, when the data item size gets larger, they constitute higher LCH and GCH ratios, as shown in Figures 5.8(d), 5.9(d) and 5.10(d) for LCH ratio, and Figures 5.8(e), 5.9(e) and 5.10(e) for GCH ratio. This is because the storage overheads of cache signatures and access history signatures become more insignificant, as the data item size gets larger. In summary, DGCC improves access latency and server request ratio in the pull-based environment, but it incurs much more power consumption than the other schemes. In the push-based and hybrid environments, the effective cache size is primarily factor to system performance. The MHs adopting DGCC achieve the best performance in terms of access latency, power consumption and server request ratio in the push-based and hybrid environments, as DGCC effectively improves the data availability in TCGs that leads to a larger effective cache size.

5.6.3 Effect of Skewness in Access Pattern

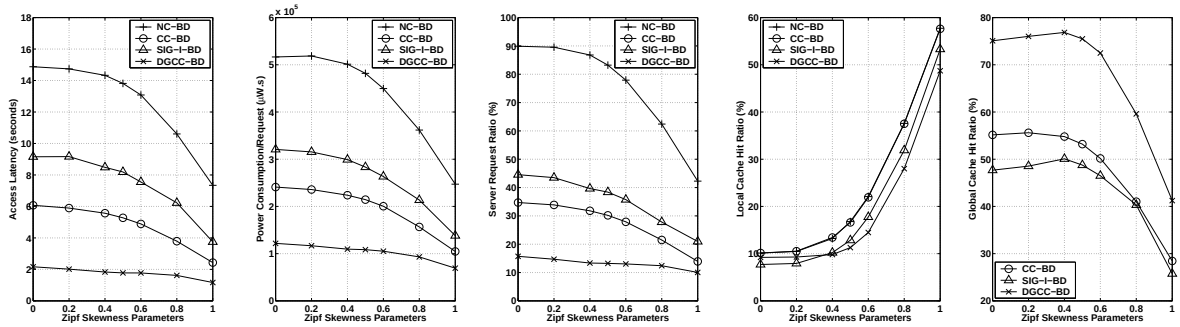


(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.11: Effect of skewness in access pattern in a pure pull-based environment.

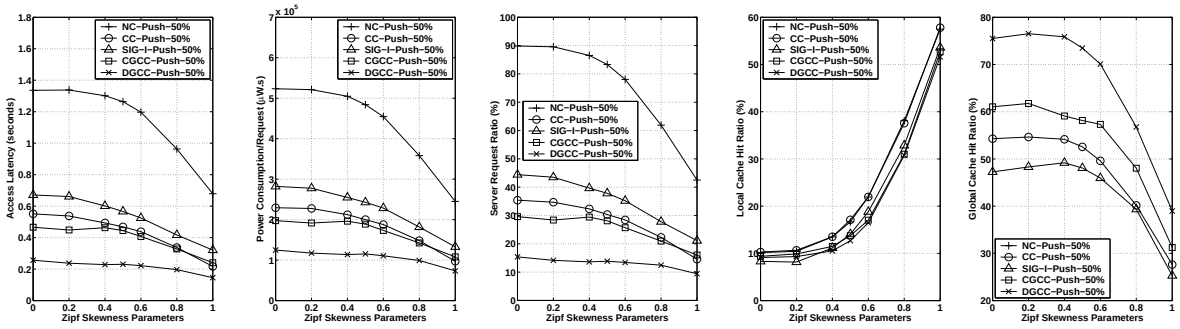
We study the effect of skewness in access pattern on system performance by varying the Zipfian skewness value from zero to one.

Figures 5.11, 5.12 and 5.13 show that the performance improves with increasing skewness parameter values in all mobile environments. The access pattern becomes



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.12: Effect of skewness in access pattern in a pure push-based environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.13: Effect of skewness in access pattern in a hybrid environment.

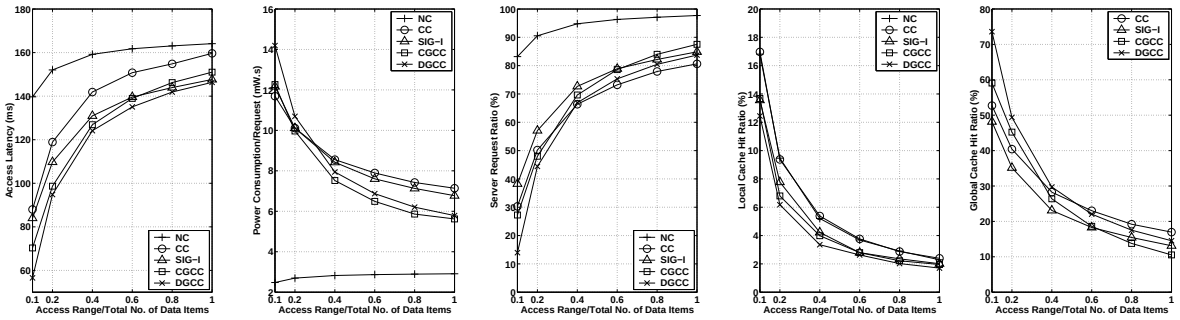
more skewed as the skewness parameter value gets larger, so there is a higher probability for MHs to obtain the required data items from the local cache, as shown in Figures 5.11(d), 5.12(d) and 5.13(d). The GCH ratio of all schemes decreases with increasing skewness value because the higher LCH ratio eases the demand for the global cache.

In the pull-based environment, DGCC outperforms other schemes in terms of access latency and server request ratio, but the MHs adopting DGCC consume more power than the other schemes because the improvement comes from a higher GCH ratio, as shown in Figure 5.11(e). Therefore, although the MHs enjoy a higher GCH ratio, they have to spend more power in handling global cache queries and exchanging cache

signatures to achieve shorter access latency.

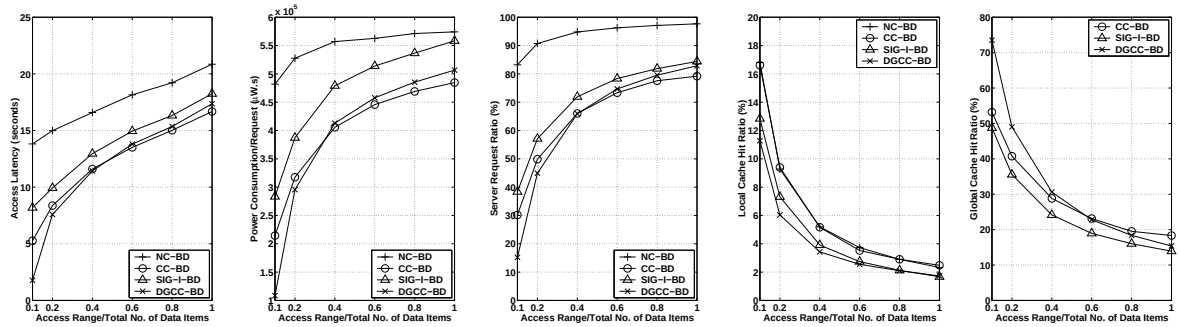
On the contrary, DGCC consistently outperforms all other schemes, in terms of access latency, server request ratio and power consumption in the push-based and hybrid environments. As the MHs adopting DGCC record the highest GCH ratio, as depicted in Figures 5.12(e) and 5.13(e), they can enjoy better system performance, it is due to the fact that a broadcast channel access incurs longer access latency and higher power consumption than a global cache access.

5.6.4 Effect of Access Density



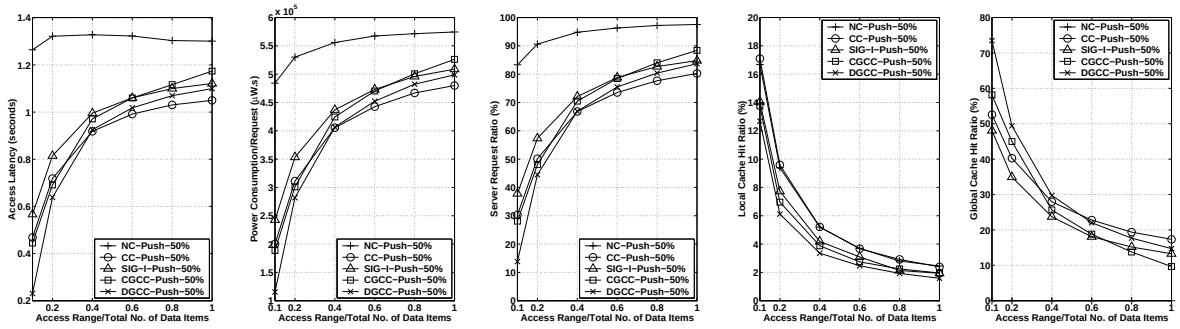
(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.14: Effect of access density in a pure pull-based environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.15: Effect of access density in a pure push-based environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.16: Effect of access density in a hybrid environment.

We study the effect of access density on system performance by increasing the access range from 1000 to 10000.

The simulation result shows that the access latency and server request ratio get worse with decreasing access density. When MHs have a larger access range, the chance of finding the required data items from the local cache decreases. Thus, the LCH ratio reduces as the access density gets lower, as shown in Figures 5.14(d), 5.15(d) and 5.16(d). Likewise, the lower access density also reduces the probability of the MHs getting required data items from their peers, so they suffer from a lower GCH ratio, as shown in Figures 5.14(e), 5.15(e) and 5.16(e).

Since the power consumption of getting data items from the MSS is much lower than getting them from the peers in the pull-based environment, the MHs can reduce power consumption when they get more data items from the MSS, as depicted as Figure 5.14(b). Thus, the power consumption of all COCA schemes reduces as the GCH ratio drops with decreasing access density. On the other hand, the power consumption of retrieving data items from the MSS is much higher than obtaining them from the peers in the push-based and hybrid environments. Figures 5.15(b) and 5.16(b) show that the MHs have to consume much more power with decreasing density in data access pattern because they suffer from a lower GCH ratio, as exhibited

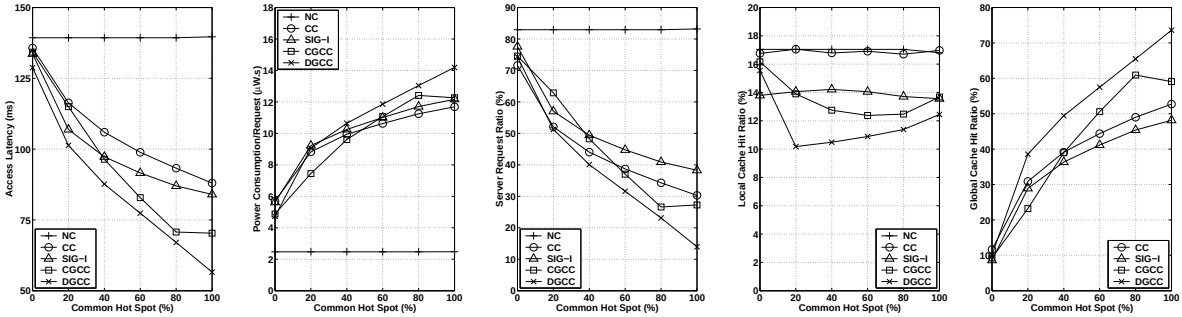
in Figures 5.15(e) and 5.16(e), so that they need to grab more desired data items from the MSS.

The MHs adopting the group-based COCA schemes suffer from a lower GCH ratio than CC in the environment with high access density. The group-based COCA schemes are dedicated to improve the data accessibility by caching more distinct data items in the aggregate cache of TCGs. The higher the access density, the larger the range of data items an MH would access. As the data access pattern is skewed, some data items may possess low access probabilities. In the group-based cache management scheme, it first keeps all distinct data items in the aggregate cache, and evicts them if the data items have not been accessed for a predefined number of replacement cycles. When the access density is low, CGCC and DGCC tend to cache many data items with low access probabilities and no replica in a TCG that is likely to reduce the GCH ratio. Thus, the performance of group-based COCA schemes may perform worse than COCA when the MHs possess low access density. DGCC can tolerate a lower data access density than CGCC. The MHs adopting DGCC are more effective in discovering tightly-coupled peers, so they can enlist more peers to help when encountering local cache misses.

5.6.5 Effect of Common Hot Spot

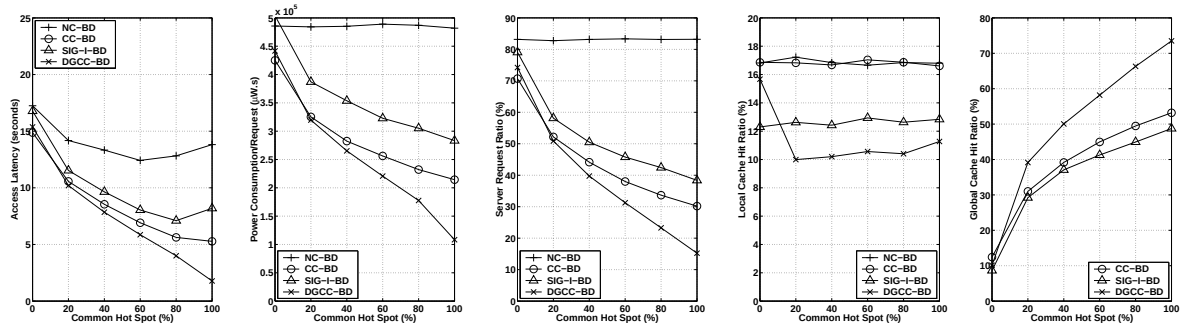
We also evaluate system performance by varying the percentage of common hot spot of the MHs in a motion group from 0 to 100 percent. The MHs in a motion group possess random access ranges when the percentage of common hot spot is zero percent. On the other hand, all MHs in the same motion group share a common access range, when the percentage of common hot spot is 100 percent.

Figures 5.17(a) and 5.17(c) show that the access latency and server request ratio of all COCA schemes get better when the percentage of common hot spot increases in



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.17: Effect of common hot spot in a pure pull-based environment.

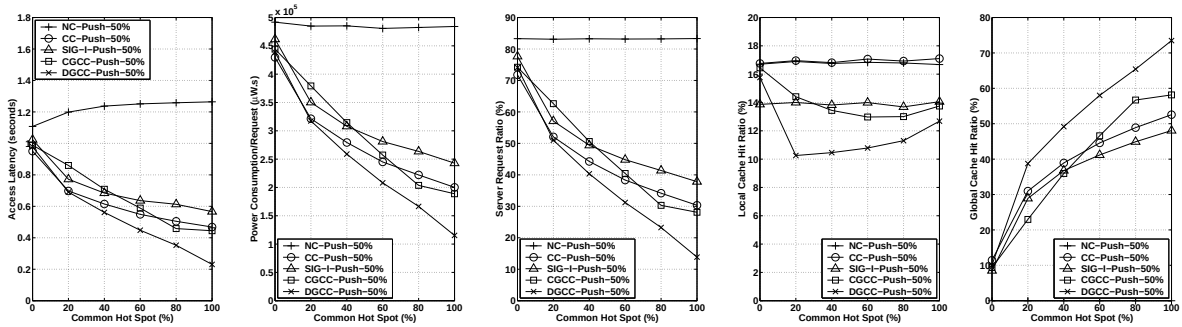


(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.18: Effect of common hot spot in a pure push-based environment.

the pull-based environment. This is because there is a higher chance for the MHs to obtain the required data items from their peers when they share a higher similarity in hot spot. The performance of DGCC is better than other schemes, as DGCC effectively improves the data accessibility in the TCGs that leads to a higher GCH ratio, as shown in Figure 5.17(e). However, the power consumption of all COCA schemes gets higher with increasing percentage of common hot spot within a motion group. When the GCH ratio increases with a higher percentage of common hot spot, the MHs have to consume more power to turn in the required data items to the requesting peers and discard more unintended messages.

In the push-based and hybrid environments, the power consumption of the MHs adopting COCA schemes reduces with increasing percentage of common hot spot, as



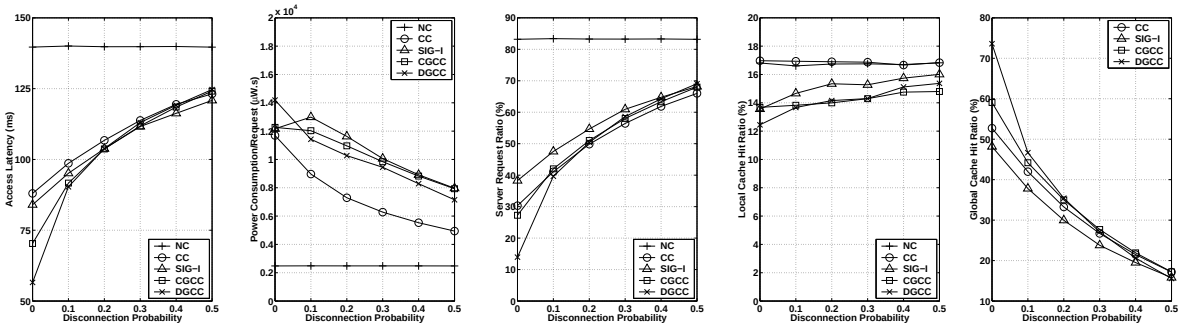
(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.19: Effect of common hot spot in a hybrid environment.

shown in Figures 5.18(b) and 5.19(b). There is a higher chance for the MHs to obtain their desired data items from the peers, so that they can save on more power, it is due to the fact that the power consumption of accessing data items from the cache of peers is less than the MSS in the push-based and hybrid environments. When the percentage of common hot spot is zero, i.e., the MHs in a motion group possess random access pattern, CC performs better than SIG-I and the group-based COCA schemes in terms of access latency, as depicted in Figure 5.18(a) for the push-based environment, and Figure 5.19(a) for the hybrid environment. This is because the MHs experience difficulty in discovering tightly-coupled peers as the score of data access similarity is not higher than the required thresholds, denoted as δ_c and δ_d for CGCC and DGCC respectively. When the percentage of common hot spot is larger than zero, the MHs can more readily discover tightly-coupled peers, so they exchange their cache signatures among the TCG members, and a portion of the cache space is then occupied by the cache signatures. Also, DGCC and CGCC try to cache more distinct data items in the global cache, when the percentage of common hot spot is low, many cached data items without any replica may possess low access probabilities. The storage of cache signatures and low percentage of common hot spot leads to a lower LCH ratio. When the percentage of common hot spot increases, the MHs in

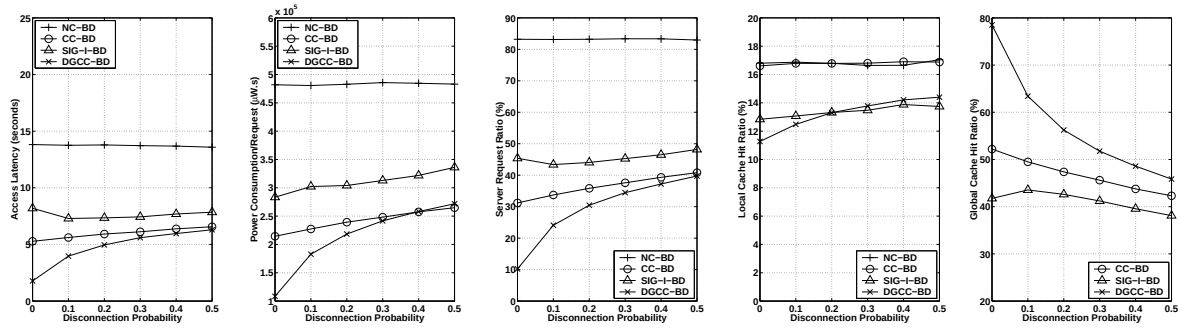
the same TCG access fewer distinct data items, so that more data items with higher access probabilities are cached in the global cache; thus, the LCH and GCH ratios improve, as exhibited in Figures 5.17(d) and 5.17(e) for the pull-based environment, Figures 5.18(d) and 5.18(e) for the push-based environment and Figures 5.19(d) and 5.19(e) for the hybrid environment.

5.6.6 Effect of Client Disconnection Probability



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

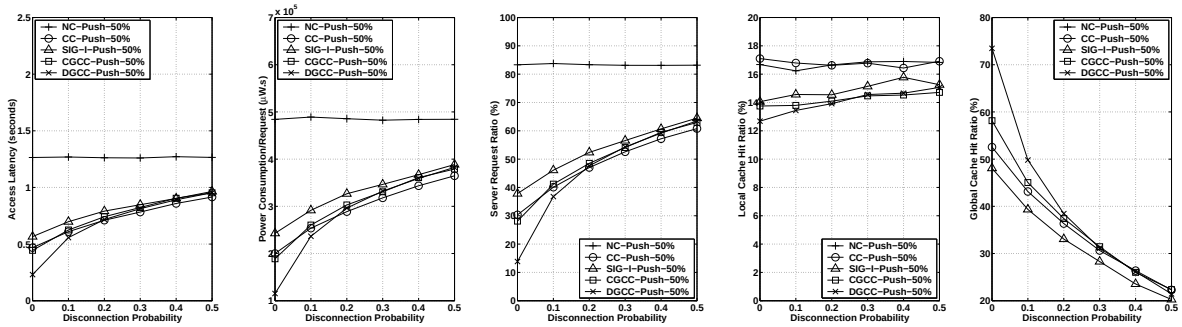
Figure 5.20: Effect of client disconnection probability in a pure pull-based environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.21: Effect of client disconnection probability in a pure push-based environment.

The effect of client disconnection on system performance is studied by increasing the client disconnection probability from 0 to 0.5.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.22: Effect of client disconnection probability in a hybrid environment.

As depicted in Figure 5.20, the performance of COCA schemes is sensitive to the disconnection rate in the pull-based environment. The performance of COCA schemes degrades with increasing disconnection probability. The higher the disconnection probability, the less the peers can help the requesting MHs. Thus, the MHs suffer from a higher global cache miss ratio, as shown in Figure 5.20(e). However, the power consumption of all COCA schemes decreases with increasing disconnection probability, as shown in Figure 5.20(b). The MHs can save on power consumption when they disconnect from the network, as they need not handle global cache queries, and discard fewer unintended messages.

DGCC performs worse than the other COCA schemes in the environment with high client disconnection probability, i.e., over 0.2. The MHs adopting DGCC can only restore the relationship with other peers after re-connecting to the network if the peers are connected to the network. Otherwise, the MHs cannot maintain the relationship. When the client disconnection probability is high, there is a higher chance for the case that the peers are disconnecting from the network when the MHs re-connect to the network. Such mutual disconnection problem degrades the system performance of DGCC. To alleviate this problem, when a re-connected MH restores the membership relationship with its peers, if it cannot communicate with

some peers, it stores the relevant relationship information. When the MH receives a relationship restore request message from them, it sends the corresponding stored relationship to them, and then they both restore the relationship. For CGCC, when there is a high disconnection probability for MHs, the location precision degrades because disconnected the MHs cannot send their location information to the MSS. With the outdated location information, the MSS cannot correctly discover TCGs, so that the system performance is also degraded as in DGCC.

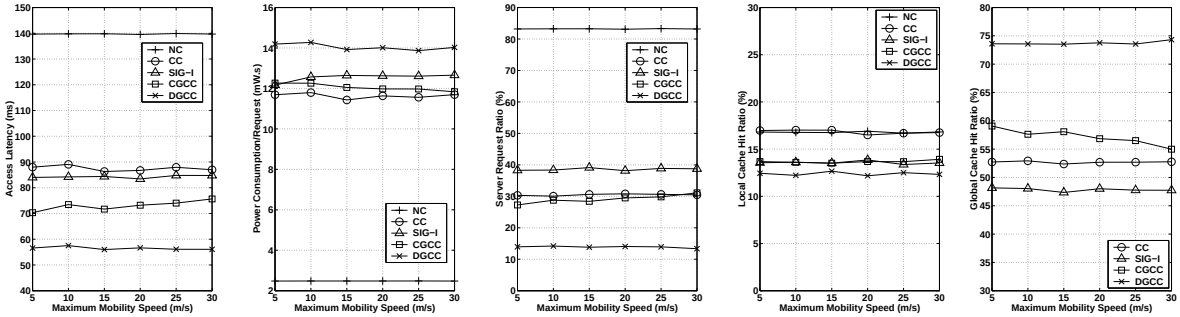
In the push-based environment, the performance of DGCC is better than other schemes. The MHs suffer from a longer latency of accessing their required data items from the MSS, as shown in Figure 5.21(a) and they need to keep themselves connected to the network to wait for the broadcast index or the data items, so the time interval between two consecutive disconnection in the push-based environment is longer than pull-based environment.

In the hybrid environment, the latency of accessing data items from the MSS is much shorter than the push-based environment, as depicted in Figure 5.22(a). Thus, the group-based COCA performs worse than CC in the environment with high client disconnection probability as in the pull-based environment.

5.6.7 Effect of Mobility Speed

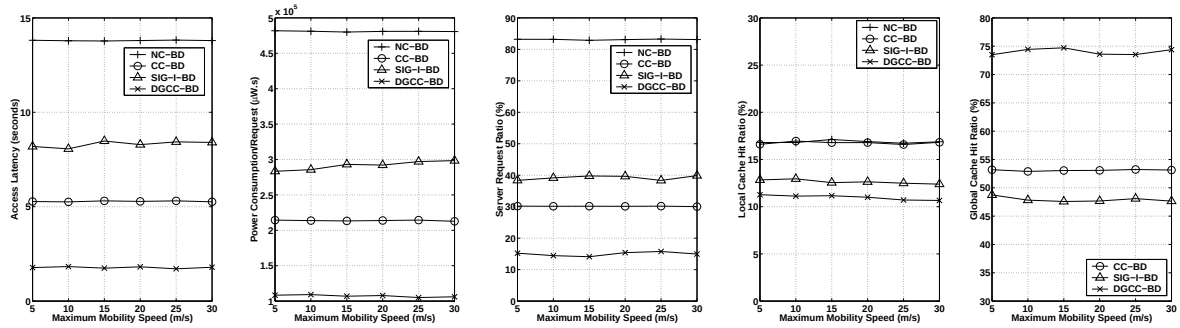
In this experiment, we study the effect of mobility speed on system performance by increasing the maximum mobility speed from 5 m/s to 30 m/s.

In the pull-based environment, CC, SIG-I and DGCC are slightly affected by varying the maximum mobility speed. However, CGCC is more sensitive to the various mobility speed than the other schemes. When the mobility speed increases, the GCH ratio of CGCC aggravates, as shown in Figure 5.23(e). As the MHs adopting CGCC only update their location by piggybacking the location information on the request



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

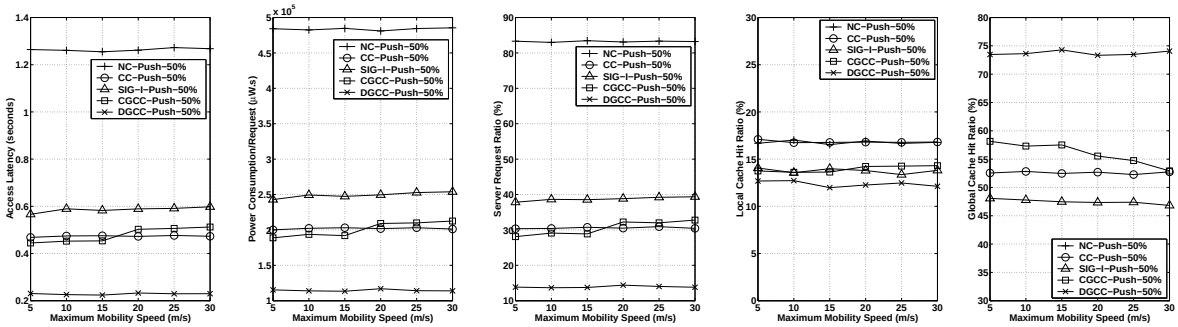
Figure 5.23: Effect of mobility speed in a pure pull-based environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.24: Effect of mobility speed in a pure push-based environment.

sent to the MSS, the high speed movement leads to imprecise location information, so that the centralized approach encounters a problem to correctly cluster the tightly-coupled peers together. For instance, two MHs, m_1 and m_2 are in fact tightly-coupled peers in the system. When m_1 encounters a cache miss, it forwards a request to the MSS along with its location information. After a few seconds, e.g., five seconds, m_2 also encounters a cache miss, it sends a request to the MSS and m_1 has not sent any request to the MSS during this time period. If m_1 and m_2 are moving with the maximum speed 30 m/s, the MSS may find that the distance between m_1 and m_2 is about 150 meters, but m_1 and m_2 are actually close to each other with only a distance of a few meters. Therefore, the centralized approach cannot tolerate the high client movement speed, unless the MHs update their location more frequently,



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.25: Effect of mobility speed in a hybrid environment.

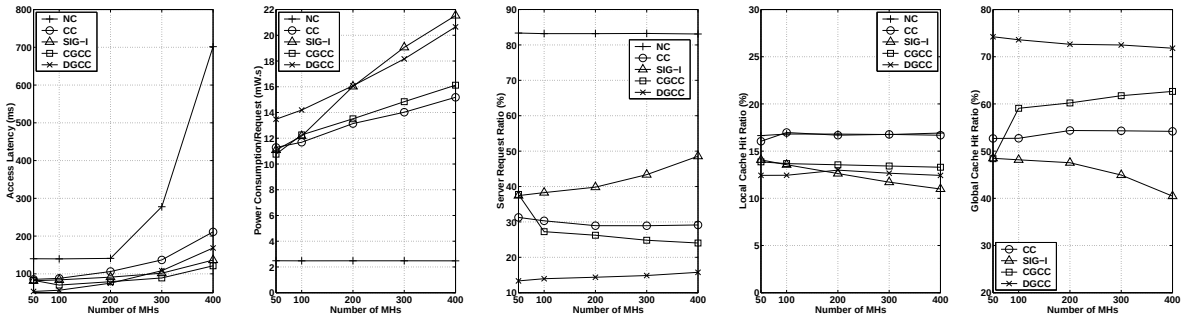
but the frequent location update could degrade system performance by generating a lot of network traffic in the uplink channel. The result in Figure 5.23 exhibits that the distributed approach, DGCC, is more adaptive to the high speed movement.

In the push-based and hybrid environments, CC, SIG-I and DGCC are also well-adapted to the various speed levels. In the hybrid environment, the CGCC performs worse with accelerated movement speed, as depicted in Figure 5.25. The MHs do not send any location information to the MSS if they can catch their desired data items in the broadcast channel, so that the knowledge of their actual location becomes more inaccurate than in the pull-based environment. Thus, the performance of CGCC in access latency, power consumption, server request ratio and GCH ratio is worse than CC, as the maximum speed is higher than 15 m/s.

5.6.8 Effect of Number of MHs

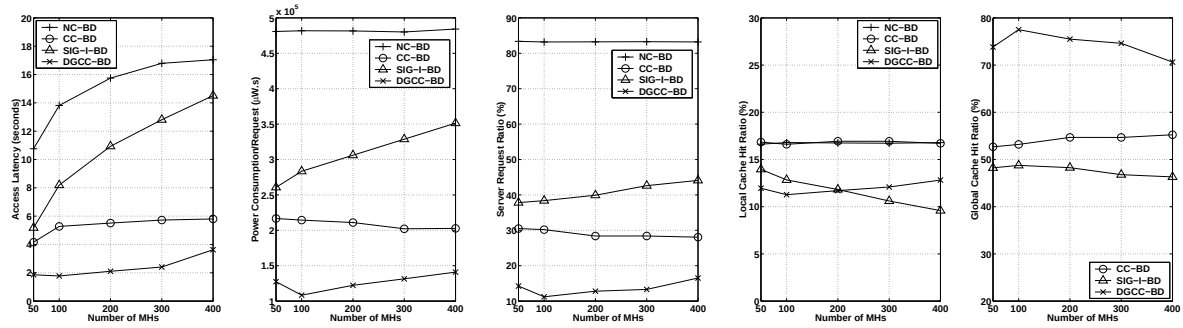
We study the effect of client population on system performance by increasing the number of MHs in the system from 50 to 400.

In the pull-based environment, all schemes are shown to perform worse with increasing client population, as shown in Figure 5.26. Figure 5.26(a) exhibits that the access latency of NC increases sharply, when there are more than 200 MHs in the



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

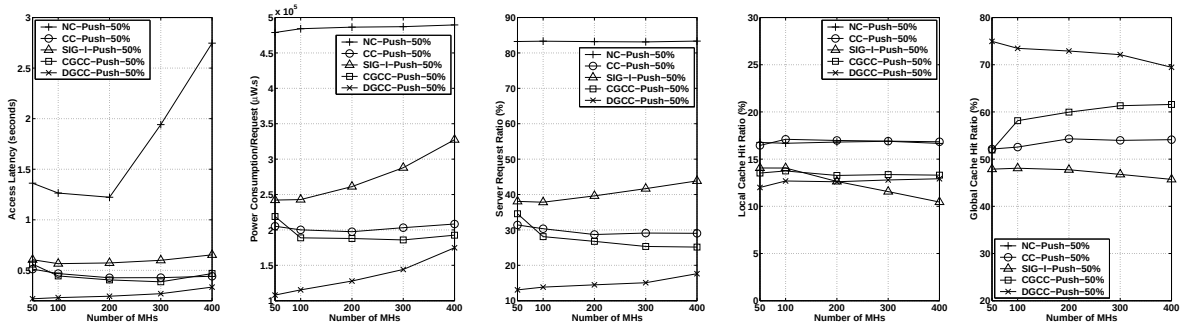
Figure 5.26: Effect of number of MHs in a pure pull-based environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.27: Effect of number of MHs in a pure push-based environment.

system. The power consumption of the COCA schemes increases, as the number of MHs gets larger. Since the access range of each motion group is randomly assigned, the increasing number of MHs does not bring in improvement in the GCH ratio as anticipated, but may even degrade the GCH ratio, as shown in Figure 5.26(e). When one motion group meets with another group, since their access ranges may not overlap with each other, they cannot share cached data items with one another. If two motion groups with no common access range meet together, they cannot take advantage of the cached data items with one another; they actually degrade the system performance in terms of access latency and power consumption. This is because they have to consume more power not only to receive more broadcast requests from the peers of another motion groups, but also to discard more unintended messages. Thus, the



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.28: Effect of number of MHs in a hybrid environment.

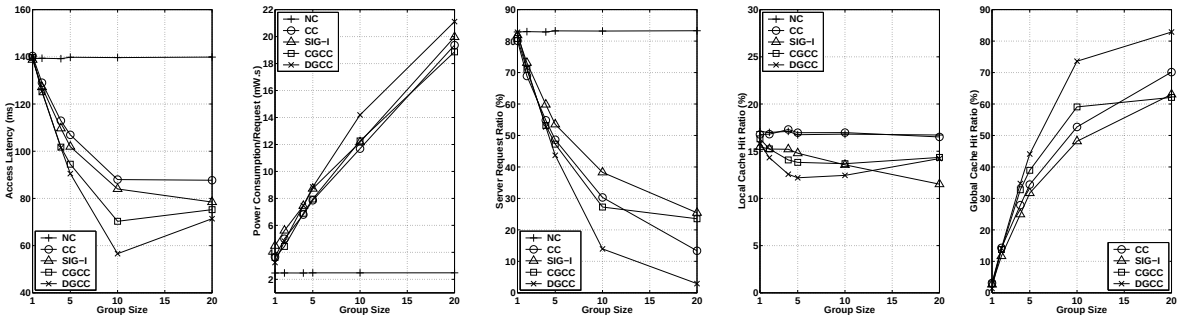
MHs adopting all the COCA schemes have to consume more power with increasing client population, as exhibited in Figure 5.26(b). For SIG-I, the MHs consume more power than the other COCA schemes, when there are more than 200 MHs in the system. As the MHs adopting SIG-I have to exchange their cache signatures with all neighboring peers, they use more cache space to store the cache signatures with increasing number of MHs, and therefore, they suffer from a decay of the LCH and GCH ratios when the number of MHs gets larger. Also, they have to consume more power on exchanging and maintaining cache signatures than DGCC and CGCC.

In the push-based environment, CC and DGCC are effective in improving access latency and reducing power consumption in comparison with NC because the MHs adopting CC and DGCC record a higher GCH ratio than NC, as shown in Figures 5.27(a), 5.27(b) and 5.27(e). For SIG-I, the access latency and power consumption get worse with increasing client population because the higher client population the more cache signature exchange an MH has to perform. Therefore, the system performance of SIG-I degrades with increasing number of MHs. As the MHs adopting DGCC only exchange their cache signatures with other tightly-coupled peers, the amount of message passing generated by signature exchange is significantly reduced. Also, the MHs adopting SIG-I suffer from lower LCH and GCH ratios, as exhibited

in Figure 5.27(d) and 5.27(e). The portion of cache space occupied by storing cache signatures is proportional to the number of neighboring peers to an MH. Thus, they use up more cache space to store cache signatures, when the number of MHs gets larger.

In the hybrid environment, we can draw similar conclusions as in the push-based environment. The COCA schemes are found to effectively improve system performance compared with NC, as illustrated in Figure 5.28. DGCC outperforms the other COCA schemes in terms of access latency, power consumption and server request ratio. Due to the overhead of exchanging cache signatures among MHs, the performance of SIG-I and DGCC degrades with increasing number of MHs. DGCC outperforms SIG-I because it only allows the MHs to exchange their cache signatures with tightly-coupled peers. For DGCC, one possible way to prevent from degrading system performance with increasing client population is to adopt transmission power control to reduce the power of a wireless NIC, so as to reduce the number of peers an MH can contact with.

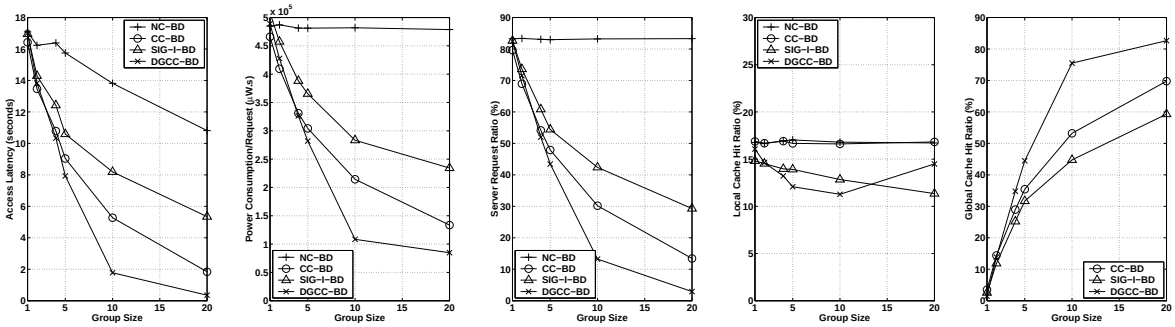
5.6.9 Effect of Group Size



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

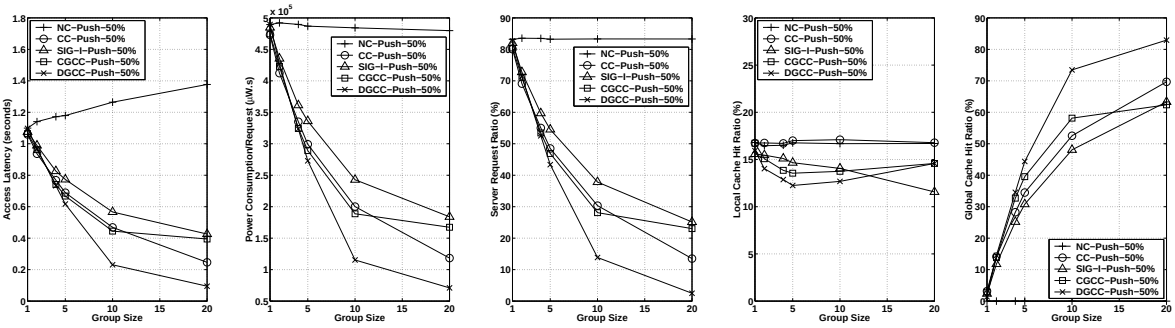
Figure 5.29: Effect of group size in a pull-based environment.

Finally, we study the effect of motion group size on system performance by increas-



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.30: Effect of group size in a push-based environment.



(a) Access Latency **(b)** Power Consumption **(c)** Server Request Ratio **(d)** LCH Ratio **(e)** GCH Ratio

Figure 5.31: Effect of group size in a hybrid environment.

ing the group size from 1 to 20. Note that when the group size is equal to one, the mobility model is equivalent to an individual random walk model, i.e., the "random waypoint" model [17, 18], and the data access range of all MHs are randomly selected.

In all the environments, the MHs adopting the COCA schemes record the lowest GCH ratio when the group size is equal to one, as shown in Figures 5.29(e), 5.30(e) and 5.31(e), that constitutes the worst case of all COCA schemes in terms of access latency and server request ratio.

In the pull-based environment, the performance of the COCA schemes is better in access latency and server request ratio with increasing group size, as depicted in Figures 5.30(a) and 5.30(c). This is because there is a higher chance for the MHs to obtain their desired data items from the peers, i.e., a higher GCH ratio, when

they have more neighboring peers with similar data affinity. Although the MHs enjoy shorter access latency, they have to consume more power to handle global cache queries, as exhibited in Figure 5.29(b). The larger group size also implies a higher GCH ratio. For every GCH, a required data item is forwarded from the source MH to the requesting MH. Thus, there are more data items passing among the MHs of the same motion group that leads to a higher network traffic around the vicinity of the motion group, which in turn increases the latency of a global cache accesses (when the motion group size is larger than 10), as shown in Figure 5.29(a).

In the push-based environment, the access latency of all schemes improves with increasing the group size, as shown in Figure 5.30(a). As the group size gets larger, there are fewer motion groups in the system. Also, each motion group shares the same access range. The fewer the number of motion groups, the smaller the range of hot spot of the system, so that the more hot data items can be allocated in the broadcast disks with faster spinning speed as number of motion groups decreases. Hence, the access latency of all schemes improves with increasing motion group size. When the motion group size increases, there is a higher chance for the MHs adopting the COCA schemes to obtain their desired data items from the peers, so they achieve a higher GCH ratio, as depicted in Figure 5.30(e). When the MHs record a higher GCH ratio, they save more power, due to the fact that the power consumption of a global cache access is much less than a broadcast channel access in the push-based environment. In the COCA schemes, DGCC is always found to perform the best in the GCH ratio, so that it can save more power than the other COCA schemes, as exhibited in Figure 5.30(b).

In the hybrid environment, the access latency of NC gets larger with increasing group size, as shown in Figure 5.31(a). As the group size increases, there are fewer motion groups in the system, so that more hot data items are allocated to the broad-

cast channel. The latency of accessing data items from the broadcast channel is longer than the pull-based point-to-point. Thus, the MHs adopting NC suffer from a longer access latency, when the motion group size increases. On the other hand, the access latency of all the COCA schemes get better with increasing group size. This is because the MHs adopting the COCA schemes can take advantage of a higher GCH ratio, to improve system performance in terms of access latency, power consumption and server request ratio, as shown in Figure 5.31. DGCC is also shown to perform the best in the GCH ratio, so the MHs adopting DGCC enjoy the best system performance.

5.7 Concluding Remarks

In this chapter, we propose two group-based schemes for MHs: *centralized* and *distributed* group-based COCA schemes, namely CGCoca and DGCoca, respectively. In the group-based COCA schemes, a collection of MHs possesses similar mobility pattern and data affinity form a group, called *tightly-coupled group* (TCG). A centralized incremental clustering algorithm is proposed for CGCoca to discover all TCGs dynamically in the system, while a distributed memorization-based membership algorithm is proposed for DGCoca. In a TCG, the cache of all MHs forms an aggregate cache or global cache. Two cooperative cache management protocols: *cooperative cache admission control* and *cooperative cache replacement* are proposed for them to work together to manage their aggregate cache. The MHs adopt the cache signature scheme not only to determine whether their desired data items are likely cached in the aggregate cache, but also to perform cooperative cache replacement protocol to improve data accessibility in the TCG. The cooperative cache admission control protocol is proposed for the MHs to control data replicas in a TCG. The performance of

CGCoca and DGCoca is extensively evaluated through a series of simulated experiments. The result shows that the group-based COCA schemes further improve system performance in comparison with standard COCA and COCA with cache signature scheme in the pull-based, push-based and hybrid environments. DGCoca gives the best in access latency and server request ratio, but incurs higher power consumption in the pull-based environment. DGCoca outperforms all other schemes in the push-based and hybrid environments. Based on the simulation results, we suggest that the distributed approach is more suitable than centralized one in discovering TCGs in mobile environments. However, there is a drawback of the group-based COCA schemes. CGCoca and DGCoca cannot tolerate a very high client disconnection probability. In a mobile environment with very frequent client disconnection, the performance of CGCoca and DGCoca is reduced to be close to the standard COCA. To alleviate the client disconnection problem, the MHs adopting DGCoca have to store the relationship information of their peers for a longer time, whereas a precise location prediction algorithm is required in CGCoca to predict the location of disconnected MHs.

Chapter 6

Conclusion

In this chapter, we will give some concluding remarks to this thesis, and then propose some potential future works for extending COCA.

6.1 Concluding Remarks of the Thesis

In this thesis, we have proposed a cooperative caching scheme, called COCA, for MHs in the pull-based, push-based and hybrid mobile environments. In COCA, the MHs retrieve their desired data items not only from the MSS, but also from the cache of their peers. COCA is also extended to support multi-hop data searching. The performance of COCA is studied through a number of simulated experiments. The result shows that COCA reduces access latency and server request, but incurs more power consumption in the pull-based mobile environment. On the other hand, COCA improves system performance in terms of access latency, server request ratio and power consumption in the push-based and hybrid environments. For COCA with multi-hop data searching, it further improves system performance in the push-based and hybrid environments.

A cache signature scheme is proposed for MHs to provide hints for them to de-

termine whether to search for their desired data items in the Peer Cache layer. To study the tradeoff between the storage space and maintenance overhead, three signature storage schemes are proposed: *individual*, *group* and *hybrid*. For the individual signature storage scheme, the peer signatures are stored in the cache individually to conserve battery power, as the peer signature can be efficiently maintained. For the group signature storage scheme, the peer signatures are grouped together and stored in a counter vector structure. When an MH detects any link failure with its peers, it has to remove the cache signature of that peers by resetting the counter vector and re-construct the counter vector by re-collecting the cache signatures from all the remaining neighbors. Thus, the group signature scheme trades power for storage space. For the hybrid signature storage scheme, it combines the individual and group signature storage schemes by defining a time threshold. The MHs initially cache the peer signature individually, and then store the cached peer signature in a counter vector when the cache signature has been cached for the time threshold. The performance of cache signature scheme with the three different storage schemes is investigated through a series of simulated experiments. Generally, the cache signature scheme with group signature storage scheme gives the best access latency, but it incurs more power consumption than all other storage schemes and standard COCA. Although the access latency of the individual signature storage scheme is slightly worse than the other two storage scheme, it incurs lower power consumption than the other signature storage schemes and the standard COCA. The performance of the hybrid storage scheme is generally between the individual and group signature schemes.

To further improve system performance, we also propose two group-based COCA schemes: *centralized* and *distributed*, namely CGCoca and DGCoca respectively. In the group-based COCA schemes, we define a group of MHs that possess similar mobility pattern and data affinity as a *tightly-coupled group* (TCG). For CGCoca, a

centralized incremental clustering algorithm is adopted to discover all TCGs dynamically in the system. For DGCoca, a distributed memorization-based membership algorithm is proposed to discover all the TCGs in a self-organized manner without any support of the MSS. In a TCG, all cache of the members are considered as an aggregate cache or global cache. Two cooperative cache management protocols: *cooperative cache admission control* and *cooperative cache replacement*, are proposed for the MHs to manage the aggregate cache cooperatively, in order to improve data accessibility. Since the exchange of cache signatures among MHs is expensive in terms of power consumption and network traffic, the MHs merely exchange their cache signatures with other peers in the same TCG. In a TCG, since all members possess similar mobility pattern, the cached peer signatures are stable. Because of this desired property, we adopt group signature storage scheme to cache all peer signatures to reduce on cache space consumption. Additionally, the peer signature is adopted not only to provide hints for the MHs to determine whether to bypass the search in the peers' cache, but also to provide information for the MHs to perform cooperative cache replacement in a TCG. Extensive simulation experiments are conducted to evaluate the performance of CGCoca and DGCoca. The simulation result depicts that CGCoca yields a better performance than standard COCA and COCA with cache signature scheme, while DGCoca outperforms all other schemes in terms of access latency and server request ratio, but it incurs more power consumption in the pull-based environment. In the push-based and hybrid environments, DGCoca outperforms all other schemes in terms of access latency, server request ratio and power consumption. The result also demonstrates that DGCoca is more effective in discovering TCGs than CGCoca, as DGCoca is well-adapted to the environment with high client movement speed. The common drawback of the group-based COCA schemes is intolerance of client disconnection. Although we have proposed client disconnec-

tion handling mechanism for CGCoca and DGCoca, they cannot tolerate very high client disconnection probability and long disconnection duration. The simulation results show that the performance of CGCoca and DGCoca becomes close to standard COCA in the environment with high client disconnection probability. To alleviate the client disconnection problem, the MHs adopting DGCoca have to store the relationship information of their peers for a longer time, whereas a precise location prediction algorithm is required in CGCoca to predict the location of disconnected MHs.

6.2 Future Work

Several important issues and improvement related to COCA remain unexplored in this thesis. To complete this research project, we will discuss some potential future work for COCA.

6.2.1 Cache Invalidation Protocol

Currently, we have not considered data updates in COCA. Our imminent step is to study the impact of data updates and design a cache coherence strategy for COCA. There are two conventional ways to maintain client cache coherency in mobile environments. First, the MSS periodically broadcasts invalidation reports (IRs) to the mobile clients to indicate which data items become invalid [11]. Second, the MSS assigns a time-to-live to each data item, when it is sent to an MH, and the MH invalidates the cached data item with expired time-to-live [73, 91]. Recently, with the emergency of new P2P communication paradigm, cooperative cache invalidation protocol [45] is proposed for the MHs to validate their cached data items in MANETs. We will investigate the performance of the conventional and cooperative cache invalidation protocols in COCA.

6.2.2 Client Incentive Scheme

In this thesis, we assume that all MHs are willing to cooperate for the benefit of the community. However, in reality, there are many selfish MHs that only enlist other peers for help, but they never or rarely share information with others, or even deny to forward any requests or data items. In P2P networks, some distributed incentive mechanisms [64, 80] are proposed for information sharing systems, to provide fairness and incentive for the community.

We would study how the selfish MHs affect the system performance in COCA, and then examine the effectiveness of the existing P2P incentive mechanisms on mobile cooperative caching, upon considering the unique properties of mobile systems.

6.2.3 Power Conservation Protocol

Based on our simulation results, we find that COCA incurs much more power consumption than conventional caching strategy in a pull-based environment. Thus, we will examine transmission power control mechanism to reduce transmission power consumption in COCA. The basic idea of power control [5, 51] is to detect the minimum required transmission power, and then the MHs adjust the power level of their wireless NICs to communicate with the required peers, in order to conserve battery power. Also, movement prediction techniques together with transmission power control [19] are also adopted to reduce power consumption in wireless communication. When an MH wants to communicate with another peer, it predicts the movement of the peer based on its movement history. If the peer is likely to move closer to the MH in the near future, the MH postpones the communication until the peer has moved close enough to it.

Since there is no transmission power control mechanism adopted by the MHs, we will examine how power control mechanisms with other techniques, such as movement

prediction, can affect the system performance of COCA.

6.2.4 Semantic Cooperative Caching

Semantic caching is highly beneficial in a mobile environment [56, 57], by enabling queries to be self-answerable, even with limited support from peers and in the absence of server connectivity, through the association of descriptive semantics to cached data items. We consider semantic caching as an alternative of cache signatures. Cache signatures provide hints for the MHs to know about the cache content of their peers, in order to perform filtering and cooperative cache replacement. In semantic caching, the cached data items are grouped into clusters based on predicates. An MH can determine whether its cached data items are sufficient in answering a query totally or partially, or not. If the cached data items cannot completely answer the query, the MH has to obtain the required data items from the MSS. The cache signature is effective in handling individual data items, whereas semantic caching is good at manipulating a group of data items based on semantic descriptor. In the future, we would like to explore on the performance of COCA with semantic caching in a mobile database environment.

6.2.5 Utilizing the Cache Space of Low-Activity MHs

Allocating appropriate data items to the low-activity MHs in COCA is found to improve system performance [25, 27], as the high-activity MHs can take advantage of the data items stored at the low-activity MHs. In [25, 27], each MH detects its own activity state by monitoring its periodic weighted cache access rate. An MH with low weighted cache access rate is considering as a low-activity client. When an MH detects its state changing to low activity state, it reports its state to the MSS, and the MSS then allocates the appropriate data items to it.

In CGCoca, since the MSS possesses knowledge about all MHs' mobility pattern and data affinity, it would be more effective in selecting the appropriate data items to be allocated to the low-activity MH by considering the access pattern of the MH's nearby peers. Similarly, in DGCoca, the MHs have knowledge about the cache content of their TCG members based on their peer signatures. In addition, they can extract the access pattern of their members from their access history signatures. A low-activity MH could determine the appropriate data items to be allocated for its own and its members' interest based on the hints. Therefore, we will attempt to utilize the cache space of low-activity MHs in the group-based COCA schemes to see if the system performance can be further improved by allocating suitable data items in the cache of the low-activity members in a TCG.

References

[1] *Squid Web Proxy Cache*, 1996. [online] http://www.squid-cache.org.

[2] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communication environments. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 199–210, San Jose, California, USA, May 1995.

[3] S. Acharya, M. Franklin, and S. Zdonik. Balancing push and pull for data broadcast. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 183–194, Tucson, Arizona, USA, May 1997.

[4] S. Acharya and S. Muthukrishnan. Scheduling on-demand broadcasts: New metrics and algorithms. In *Proceedings of the 4th International Conference Mobile Computing and Networking (MobiCom)*, pages 43–54, Dallas, Texas, USA, October 1998.

[5] S. Agarwal, R. H. Katz, S. V. Krishnamurthy, and S. K. Dao. Distributed power control in ad-hoc wireless networks. In *Proceedings of the the 12th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 59–66, San Diego, California, USA, September 2001.

[6] D. Aksoy and M. Franklin. RxW: A scheduling approach for large-scale on-demand data broadcast. *IEEE/ACM Transactions on Networking (TON)*, 7(6):846–880, December 1999.

[7] B. An and S. Papavassiliou. A mobility-based clustering approach to support mobility management and multicast routing in mobile ad-hoc wireless networks. *International Journal of Network Management*, 11(6):387–395, November 2001.

[8] M. J. Atallah. *Algorithms and Theory of Computation Handbook*, chapter Basic Graph Algorithms. CRC Press, 1999.

[9] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrievel*. Addison-Wesley, 1999.

[10] L. R. Bahl and H. Kobayashi. Image data compression by predictive coding II: Encoding algorithms. *IBM Journal of Research and Development*, 18(2):172–179, March 1974.

[11] D. Barbará and T. Imielinski. Sleepers and workaholics: Caching strategies in mobile environments. *The International Journal on Very Large Data Bases*, 4(4):567–602, October 1995.

[12] D. Barbarh. Mobile computing and databses - a survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 11(1):108–117, January/February 1999.

[13] P. Basu, N. Khan, , and T. D. Little. A mobility based metric for clustering in mobile ad hoc networks. In *Proceedings of the International Workshop on Wireless Networks and Mobile Computing, in conjunction with the 21st International Conference on Distributed Computing Systems (ICDCS)*, pages 413–418, Phoenix, Arizona, USA, April 2001.

[14] K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems (TOCS)*, 9(3):272–314, August 1991.

[15] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM (CACM)*, 13(7):422–426, July 1970.

[16] Bluetooth SIG. *Bluetooth specification v1.2*. [online] http://www.bluetooth.com, November 2003.

[17] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 85–97, Dallas, Texas, USA, October 1998.

[18] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing (WCMC)*, 2(5):483–502, August 2002.

[19] S. Chakraborty, Y. Dong, D. K. Y. Yau, and J. C. S. Liu. On the effectiveness of movement prediction to reduce energy consumption in wireless communication. *IEEE Transactions on Mobile Computing (TMC)*, to appear.

[20] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical internet object cache. In *Proceedings of the USENIX Annual Technical Conference*, pages 153–164, San Diego, California, USA, January 1996.

[21] J. H. P. Chim, M. Green, R. W. H. Lau, H. V. Leong, and A. Si. On caching and prefetching of virtual objects in distributed virtual environments. In *Proceedings of the 6th ACM International Conference on Multimedia*, pages 171–180, Bristol, England, September 1998.

[22] C.-Y. Chow, H. V. Leong, and A. T. S. Chan. Cache signatures for peer-to-peer cooperative caching in mobile environments. In *Proceedings of the 18th International Conference on Advanced Information Networking and Applications (AINA)*, pages 96–101, Fukuoka, Japan, March 2004.

[23] C.-Y. Chow, H. V. Leong, and A. T. S. Chan. Group-based cooperative cache management for mobile clients in a mobile environment. In *Proceedings of the 33rd International Conference on Parallel Processing (ICPP)*, pages 83–90, Montreal, Canada, August 2004.

[24] C.-Y. Chow, H. V. Leong, and A. T. S. Chan. Peer-to-peer cooperative caching in a hybrid data delivery environment. In *Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN)*, pages 79–84, Hong Kong, May 2004.

[25] C.-Y. Chow, H. V. Leong, and A. T. S. Chan. Peer-to-peer cooperative caching in mobile environments. In *Proceedings of the 2nd IEEE International Workshop on Mobile Distributed Computing (MDC), in conjunction with the 24th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 528–533, Tokyo, Japan, March 2004.

[26] C.-Y. Chow, H. V. Leong, and A. T. S. Chan. Distributed group-based cooperative caching in a mobile broadcast environment. In *Proceedings of the Sixth International Conference on Mobile Data Management (MDM)*, pages 97–106, Ayia Napa, Cyprus, May 2005.

[27] C.-Y. Chow, H. V. Leong, and A. T. S. Chan. Utilizing the cache space of low-activity clients in a mobile cooperative caching environment. *International Journal of Wireless and Mobile Computing (IJWMC)*, to appear.

[28] M. Dahlin, T. Anderson, D. Patterson, and R. Wang. Cooperative caching: Using remote client memory to improve file system performance. In *Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 267–280, Monterey, California, USA, November 1994.

[29] A. Ephremides, J. Wieselthier, and D. J. Baker. A design concept for reliable mobile radio networks with frequency hopping signaling. *Proceedings of IEEE*, 75(1):56–73, January 1987.

[30] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking (TON)*, 8(3):281–293, June 2000.

[31] M. Feeley, W. Morgan, E. Pighin, A. Karlin, H. Levy, and C. Thekkath. Implementing global memory management in a workstation cluster. In *Proceedings of the 15th ACM Symposium on Operating Systems Principle (SOSP)*, pages 201–212, Colorado, USA, December 1995.

[32] L. M. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1548–1557, Anchorage, Alaska, USA, April 2001.

[33] L. D. Fife and L. Gruenwald. Research issues for data communication in mobile ad-hoc network database systems. *ACM SIGMOD Record*, 32(2):42–47, June 2003.

[34] Z. Genova and K. Christensen. Using signatures to improve URL routing. In *Proceedings of the 21st IEEE International Performance Computing and Com-*

munications Conference (IPCCC), pages 45–52, Phoenix, Arizona, USA, April 2002.

[35] M. Gerla and J. T.-C. Tsai. Multicluster, mobile, multimedia radio network. *Wireless Networks (WINET)*, 1(3):255–265, January 1995.

[36] I. A. Getting. The global positioning system. *IEEE Spectrum*, 12(30):36–38, 43–4, December 1993.

[37] Y. Guo, M. C. Pinotti, and S. K. Das. A new hybrid broadcast scheduling algorithm for asymmetric communication systems. *IEEE Transactions on Computers*, 5(3):39–54, July 2001.

[38] Z. J. Haas. The routing algorithm for the reconfigurable wireless networks. In *Proceedings of IEEE 6th International Conference on Universal Personal Communications (ICUPC)*, pages 562–566, San Diego, California, USA, October 1997.

[39] Z. J. Haas and M. R. Pearlman. The performance of query control schemes for the zone routing protocol. *IEEE/ACM Transactions on Networking (TON)*, 9(4):427–438, August 2001.

[40] T. Hara. Effective replica allocation in ad hoc networks for improving data accessibility. In *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1568–1576, Anchorage, Alaska, USA, April 2001.

[41] T. Hara. Cooperative caching by mobile clients in push-based information systems. In *Proceedings of the 11th International Conference on Information and Knowledge Management (CIKM)*, pages 186–193, McLean, Virginia, USA, November 2002.

[42] T. Hara. Replica allocation in ad hoc networks with periodic data update. In *Proceedings of the 3rd Interntional Conference on Mobile Data Management (MDM)*, pages 79–86, Singapore, January 2002.

[43] T. Hara, Y.-H. Loh, and S. Nishio. Data replication methods based on the stability of radio links in ad hoc networks. In *Proceedings of the 6th International Workshop on Mobility in Databases and Distributed Systems (MDDS), in conjunction with the 14th International Conference on Database and Expert Systems Applications (DEXA)*, pages 969–973, Prague, Czech Republic, September 2003.

[44] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The anatomy of a context-aware application. *Wireless Network (WINET)*, 8(2-3):187–197, March-May 2002.

[45] H. Hayashi, T. Hara, and S. Nishio. Cache invalidation for updated data in ad hoc networks. In *Proceedings of the 11st International Conference on Cooperative Information Systems (CoopIS)*, pages 516–535, Catania, Sicily, Italy, November 2003.

[46] X. Hong, M. Gerla, G. Pei, and C.-C. Chiang. A group mobility model for ad hoc wireless networks. In *Proceedings of the 2nd International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, pages 53–60, Seattle, Washington, USA, August 1999.

[47] Q. Hu, D. L. Lee, and W.-C. Lee. Performance evaluation of a wireless hierarchical data dissemination system. In *Proceedings of the 5th International Conference Mobile Computing and Networking (MobiCom)*, pages 163–173, Seattle, Washington, USA, August 1999.

[48] J.-L. Huang, M.-S. Chen, and W.-C. Peng. Exploring group mobility for replica data allocation in a mobile environment. In *Proceedings of the 12th International Conference on Information and Knowledge Management (CIKM)*, pages 161–168, New Orleans, Louisiana, USA, November 2003.

[49] IEEE Std. 802-11, IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification. June 1997.

[50] T. Imielinski and B. R. Badrinath. Mobile wireless computing: challenges in data management. *Communications of the ACM (CACM)*, 37(10):18–28, October 1994.

[51] E.-S. Jung and H. H. Vaidya. A power control mac protocol for ad hoc networks. In *Proceedings of the 8th International Conference on Mobile Computing and Networking (MobiCom)*, pages 36–47, Atlanta, Georgia, USA, September 2002.

[52] G. H. K. Lam, H. V. Leong, and S. C. F. Chan. Gbl: Group-based location updating in mobile environment. In *Proceedings of the 9th International Conference on Database Systems for Advances Applications (DASFAA)*, pages 762–774, Jeju Island, Korea, March 2004.

[53] G. H. K. Lam, H. V. Leong, and S. C. F. Chan. Leadership maintenance in group-based location management scheme. In *Proceedings of the International Conference on Cooperative Information Systems (CoopIS), Part I*, pages 25–29, Agia Napa, Cyprus, October 2004.

[54] G. H. K. Lam, H. V. Leong, and S. C. F. Chan. Reducing group management overhead in group-based location management. In *Proceedings of the 7th International Workshop on Mobility in Databases and Distributed Systems (MDDS),*

in conjunction with the 15th International Conference on Database and Expert Systems Applications (DEXA), pages 640–644, Zaragoza, Spain, August 2004.

[55] W. H. O. Lau, M. Kumar, and S. Venkatesh. A cooperative cache architecture in support of caching multimedia objects in MANETs. In *Proceedings of the 5th ACM International Workshop on Wireless Mobile Multimedia (WoWMoM), in conjunction with the 8th International Conference on Mobile Computing and Networking (MobiCom)*, pages 56–63, Atlanta, Georgia, USA, September 2002.

[56] K. C. K. Lee, H. V. Leong, and A. Si. Semantic data access in an asymmetric mobile environment. In *Proceedings of the 3rd International Conference on Mobile Data Management (MDM)*, pages 94–101, Singapore, January 2002.

[57] K. C. K. Lee, H. V. Leong, and A. Si. Semantic data broadcast for a mobile environment based on dynamic and adaptive chunking. *IEEE Transactions on Computers (TOC)*, 51(10):1253–1268, October 2002.

[58] K.-W. Lee, K. Amiri, S. Sahu, and C. Venkatramani. Understanding the potential benefits of cooperation among proxies: Taxonomy and analysis. *IBM Research Report # RC22173*, September 2001.

[59] W.-C. Lee and D. L. Lee. Signature caching techniques for information filtering in mobile environments. *Wireless Networks (WINET)*, 5(1):57–67, January 1999.

[60] H. V. Leong and A. Si. Database caching over the air-storage. *The Computer Journal*, 40(7):401–415, 1997.

[61] L. Li, J. Y. Halpern, P. Bahl, Y.-M. Wang, and R. Wattenhofer. Analysis of a cone-based distributed topology control algorithm for wireless multi-hop networks. In *Proceedings of the 20th ACM symposium on Principles of Distributed Computing (PODC)*, pages 264–273, Newport, Rhode Island, USA, August 2001.

[62] M. Li, W.-C. Lee, and A. Sivasubramaniam. Neighborhood signatures for searching P2P networks. In *Proceedings of the 7th International Database Engineering and Application Symposium (IDEAS)*, pages 149–159, Hong Kong, July 2003.

[63] S. Lim, W.-C. Lee, G. Cao, and C. R. Das. A novel caching scheme for internet based mobile ad hoc networks. In *Proceedings of the 12th IEEE International Conference on Computer Communications and Networks (ICCCN)*, pages 38–43, Dallas, Texas, USA, October 2003.

[64] R. T. B. Ma, S. C. M. Lee, J. C. S. Lui, and D. K. Y. Yau. An incentive mechanism for p2p networks. In *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 516–523, Hachioji, Tokyo, Japan, March 2004.

[65] M. Mitzenmacher. Compressed bloom filters. *IEEE/ACM Transactions on Networking (TON)*, 10(5):604–612, October 2002.

[66] M. Papadopouli and H. Schulzrinne. Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices. In *Proceedings of the 2nd ACM Interational Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 117–127, Long Beach, California, USA, October 2001.

[67] A. K. Parekh. Selecting routers in ad-hoc wireless network. In *Proceedings of the SBT/IEEE International Telecommunications Symposium (ITS)*, pages 420–424, Mexico, August 1994.

[68] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket location-support system. In *Proceedings of the 6th International Conference on Mobile*

Computing and Networking (MobiCom), pages 32–43, Boston, Massachusetts, USA, August 2000.

[69] L. Ramaswamy and L. Liu. An expiration-age based document placement scheme for cooperative web caching. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 16(5):585–600, May 2004.

[70] R. Rivest. *The MD5 Message-Digest Algorithm.* Network Working Group RFC1321, April 1992.

[71] K. W. Ross. Hash routing for collections of shared web caches. *IEEE Network Magazine*, 11(6):37–44, November/December 1997.

[72] A. Rousskov and D. Wessels. Cache digests. *Computer Networks and ISDN Systems*, 30(22-23):2155–2168, November 1998.

[73] F. Sailhan and V. Issarny. Cooperative caching in ad hoc networks. In *Proceedings of the 4th International Conference on Mobile Data Management (MDM)*, pages 13–28, Melbourne, Australia, January 2003.

[74] P. Sarkar and J. H. Hartman. Hint-based cooperative caching. *ACM Transactions on Computer Systems (TOCS)*, 18(4):387–419, November 2000.

[75] H. Schwetman. *User's Guide CSIM19 Simulation Engine (C++ Version).* Mesquite Software Inc.

[76] R. Sedgewick and P. Flajolet. *An Introduction to the Analysis of Algorithms.* Addison-Wesley, 1996.

[77] H. Shen, S. K. Das, M. Kumar, and Z. Wang. Cooperative caching with optimal radius in hybrid wireless network. In *Proceedings of the 3rd International IFIP-TC6 Networking Conference*, pages 841–853, Athens, Greece, May 2004.

[78] K. Stathatos, N. Roussopoulos, and J. S. Baras. Adaptive data broadcast in hybrid networks. In *Proceedings of the 23rd International Conference Very Large Data Bases (VLDB)*, pages 326–335, Athens, Greece, August 1997.

[79] R. Stewart and Q. Xie. *Stream Control Transmission Protocol (SCTP): a reference guide*. Addison-Wesley Publishing Company, Boston, 2001.

[80] Q. Sun and H. Garcia-Molina:. Slic: A selfish link-based incentive mechanism for unstructured peer-to-peer networks. In *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 506–515, Hachioji, Tokyo, Japan, March 2004.

[81] D. G. Thaler and C. V. Ravishankar. Using name-based mappings to increase hit rates. *IEEE/ACM Transactions on Networking (TON)*, 6(1):1–14, February 1998.

[82] V. Valloppillil and K. W. Ross. *Cache Array Routing Protocol (CARP) v1.1*. Internet Draft, draft-vinod-carp-v1-03.txt, February 1998.

[83] G. Voelker, E. Anderson, T. Kimbrel, M. Feeley, J. Chase, A. Karlin, and H. Levy. Implementing cooperative prefetching and caching in a globally-managed memory system. In *Proceedings of the ACM SIGMETRICS International Conference on Measuring and Modeling of Computer Systems*, pages 33–43, Madison, Wisconsin, USA, June 1998.

[84] G. M. Voelker, H. A. Jamrozik, M. K. Vernon, H. M. Levy, and E. D. Lazowska. Managing server load in global memory systems. In *Proceedings of the ACM SIGMETRICS International Conference on Measuring and Modeling of Computer Systems*, pages 127–138, Seattle, Washington, USA, June 1997.

[85] K. H. Wang and B. Li. Efficient and guaranteed service coverage in partitionable mobile ad-hoc networks. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1089–1098, New York, USA, June 2002.

[86] D. Wessels and K. Claffy. ICP and the Squid web cache. *IEEE Journal on Selected Areas in Communications (JSAC)*, 16(3):345–357, April 1996.

[87] D. Wessels and K. Claffy. *Internet Cache Protocol (ICP) (Version 2) Application Specification.* Network Working Group RFC2187, September 1997.

[88] D. Wessels and K. Claffy. *Internet Cache Protocol (ICP) (Version 2) Specification.* Network Working Group RFC2186, September 1997.

[89] J. Wong. Broadcast delivery. *Proceedings of the IEEE*, 76(12):1566–1577, December 1988.

[90] J. Xu, B. Zheng, M. Zhu, and D. L. Lee. Research challenges in information access and dissemination in a mobile environment. In *Proceedings of the Pan-Yellow-Sea International Workshop on Information Technologies for Network Era (PYIWIT)*, pages 1–8, Saga, Japan, March 2002.

[91] L. Yin and G. Cao. Supporting cooperative caching in ad hoc networks. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 2537–2547, Hong Kong, March 2004.

[92] G. K. Zipf. *Human Behavior and the Principle of Least Effort.* Addison-Wesley, 1949.