8-2014

Towards Efficient File Sharing and Packet Routing in Mobile Opportunistic Networks

Kang Chen
Clemson University, kangc@g.clemson.edu

Towards Efficient File Sharing and Packet Routing in Mobile Opportunistic Networks

---

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Computer Engineering

---

by
Kang Chen
August 2014

Accepted by:
Dr. Haiying Shen, Committee Chair
Dr. Richard R. Brooks
Dr. Kuang-Ching (KC) Wang
Dr. James Martin

Abstract

With the increasing popularity of portable digital devices (e.g., smartphones, laptops, and tablets), mobile opportunistic networks (MONs) [40,90] consisting of portable devices have attracted much attention recently. MONs are also known as pocket switched networks (PSNs) [52]. MONs can be regarded as a special form of mobile ad hoc networks (MANETs) [7] or delay tolerant networks (DTNs) [35, 56]. In such networks, mobile nodes (devices) move continuously and meet opportunistically. Two mobile nodes can communicate with each other only when they are within the communication range of each other in a peer-to-peer (P2P) manner (i.e., without the need of infrastructures). Therefore, such a network structure can potentially provide file sharing or packet routing services among portable devices without the support of network infrastructures. On the other hand, mobile opportunistic networks often experience frequent network partition, and no end-to-end contemporaneous path can be ensured in the network. These distinctive properties make traditional file sharing or packet routing algorithms in Internet or mobile networks a formidable challenge in MONs. In summary, it is essential and important to achieve efficient file sharing and packet routing algorithms in MONs, which are the key for providing practical and novel services and applications over such networks. In this dissertation, we develop several methods to resolve the aforementioned challenges.

Firstly, we propose two methods to enhance file sharing efficiency in MONs by creating replicas and by leveraging social network properties, respectively. In the first method, we investigate how to create file replicas to optimize file availability for file sharing in MONs. We introduce a new concept of resource for file replication, which considers both node storage and meeting frequency with other nodes. We theoretically study the influence of resource allocation on the average file access delay and derive a resource allocation rule to minimize the average file access delay. We also propose a distributed file replication protocol to realize the deduced optimal file replication rule. In

the second method, we leverage social network properties to improve the file searching efficiency in MONs. This method groups common-interest nodes that frequently meet with each other into a community. It takes advantage of node mobility by designating stable nodes, which have the most frequent contact with community members, as community coordinators for intra-community file request forwarding, and highly-mobile nodes that visit other communities frequently as community ambassadors for inter-community file request forwarding. Based on such a community structure, an interest-oriented file searching scheme is proposed to first search local community and then search the community that is most likely to contain the requested file, leading to highly efficient file sharing in MONs.

Secondly, we propose two methods to realize efficient packet routing among mobile nodes and among different landmarks in MONs, respectively. The first method utilizes distributed social map to route packets to mobile nodes efficiently with a low-cost in MONs. Each node builds its own social map consisting of nodes it has met and their frequently encountered nodes in a distributed manner. Based on both encountering frequency and social closeness of two linked nodes in the social map, we decide the weight of each link to reflect the packet delivery ability between the two nodes. The social map enables more accurate forwarder selection through a broader view and reduces the cost on information exchange. The second method realizes high-throughput packet routing among different landmarks in MONs. It selects popular places that nodes visit frequently as landmarks and divides the entire MON area into sub-areas represented by landmarks. Nodes transiting between two landmarks relay packets between the two landmarks. The frequency of node transits between two landmarks is measured to represent the forwarding capacity between them, based on which routing tables are built on each landmark to guide packet routing. Finally, packets are routed landmark by landmark to reach their destination landmarks.

Extensive analysis and real-trace based experiments are conducted to support the designs in this dissertation and demonstrate the effectiveness of the proposed methods in comparison with the state-of-art methods. In the future, we plan to further enhance the file sharing and packet routing efficiency by considering more realistic scenarios or including more useful information. We will also investigate the security and privacy issues in the proposed methods.

Acknowledgments

I would like to first thank my advisor Dr. Haiying Shen for her continuous advise, support, encourage, and help during my PhD study in Clemson, without whom it is impossible for me to finish my dissertation research productively. The knowledge and spirit learned from her will be my lifelong treasure and will benefit me in my future career and life.

I am also deeply grateful to my committee members: Dr. Richard R. Brooks, Dr. Kuang-Ching Wang, and Dr. James Martin. Their professional opinions and encourages have greatly helped me to finish my dissertation research. I am also inspired by their broad knowledge, outstanding research, and the dedication to research to pursue my Ph.D. and future career.

My life in Clemson University has been joyful, fruitful, and memorable. I am grateful to all members in the Pervasive Communication Lab for their help and encourage, as well as the days and nights we studied and worked together. I have to thank all friends in Clemson, who have provided me numerous encourage and happiness. Without these labmates and friends, my experience at Clemson would not be as great as it currently is.

Finally and most importantly, I am always grateful to my parents, Youxin Chen and Manyun Li, and my brother, Le Chen, for their never-ending support and love.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the past few years, personal mobile devices such as laptops, PDAs, and smartphones have been more and more popular. Indeed, the number of smartphone users reached 1 billion by the third quarter of 2012 and is expected to increase to 2 billion by 2015 [6]. The incredibly rapid growth of mobile users is leading to a promising future, in which they can form a mobile opportunistic network (MON) [40, 90], which is also known as pocket switched network (PSNs) [52], to freely share files or forward packets between each other without the support of cellular infrastructures. Such networks are often shown in the form of mobile ad hoc networks (MANETs) [7] or delay tolerant networks (DTNs) [35, 56], in which mobile devices, carried by people, are interconnected by opportunistic encountering. Though communication infrastructures exist commonly nowadays, we focus on exploring the unused peer-to-peer (P2P) communication among digital devices (e.g., through WiFi and Bluetooth) for pervasive communication and computing without the constraint of infrastructures. In such a scenario, mobile devices establish connections for message exchange only when they are within the communication range of each other, i.e., encountering based communication. As a result, MONs often experience frequent network partitions, and no end-to-end contemporaneous path can be ensured in the network.

These distinctive properties make traditional file sharing or packet routing algorithms in Internet or mobile networks challenging in MONs. As a result, file searching and packet routing usually are realized in a "store-carry-forward" manner in MONs [56]. Specifically, when a node receives a file request or a packet, it carries the request/packet while moving in the network until meeting the destination node or a node that is more suitable to carry the request/packet. Then,

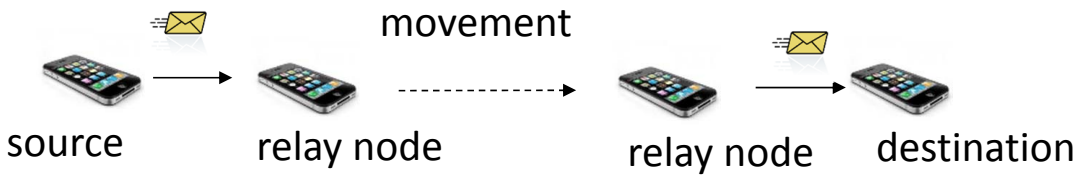


Figure 1.1: Demonstration of request/packet forwarding process in MONs.

the request/packet is forwarded to the newly met node. Through such a hop-by-hop forwarding strategy, the request/packet finally reaches the file holder/destination node. Figure 1.1 demonstrates this process. In the figure, a file request/packet is generated by the "source node", which forwards it to its nearby "relay node". However, the "relay node" fails to find a node that is more suitable to carry the request/packet from nearby nodes. Then, it carries the request/packet while moving around until it meets a more suitable "relay node" or the file holder or the destination of the packet. Then, the "relay node" forwards the request/packet to it.

The file sharing or packet routing process actually utilizes the node mobility in MONs to make up the lack of end-to-end paths. Therefore, deciding which node is more suitable to carry the request/packet is the key for efficient file sharing or packet routing in MONs. Ideally, the request/packet should be carried by the node that is most likely to meet the file holder or the destination node. Generally, when two nodes meet, they first exchange certain information to determine which node is more suitable for carrying which request/packet and then forward requests or packets to the other node accordingly. This is also the basic direction in the state-of-art MON file sharing/packet routing algorithms [9, 10, 14, 20, 22, 25, 29, 30, 36, 49, 53, 61, 62, 64, 65, 67, 68, 71–73, 80, 82, 92, 93].

1.1 Problem Statement

Despite of the aforementioned basic method that enables the relaying of file requests/packets among nodes in MONs, the limited resources on mobile nodes and the dynamism of node mobility make highly efficient file sharing or packet routing non-trivial. Therefore, we aim to solve below two challenges in this dissertation.

Firstly, how to realize efficient file sharing? By file sharing, we refer to the scenario in which nodes request files from other nodes in the system without a file server. This is because the

distributed nature of MONs makes centralized file sharing challenging in MONs. File replication is an intuitive way to enhance the file availability in MONs. However, previous methods [49,61,71,80,82] simply consider storage as the resource for replicas and neglect that a node's frequency to meet other nodes (meeting ability in short) also influences the availability of its file. Files in a node with higher meeting ability have higher availability. Then, how to consider node storage and mobility jointly for replica creation in order to achieve the optimal file availability, i.e., minimum global average file access delay, requires further investigation. Moreover, nodes in MONs usually belong to certain social networks and present certain social network properties, which can affect their mobility and encountering with each other in the network. Therefore, it would be beneficial to leverage social network properties to improve the file sharing efficiency.

Secondly, how to realize efficient packet routing among both nodes and landmarks? Current MON packet routing algorithms exploit either previous encountering records [9, 14, 64, 72] or social network properties [10, 29, 29, 30, 53, 68, 92] to deduce a node's ability to deliver a packet to its destination. However, this may lead to two drawbacks: 1) the delivery ability only reflects a node's direct encounter probability or 2-hop accumulated relay probability and thereby only provides a local view on forwarder selection; 2) two encountered nodes need to exchange the information on their delivery abilities to the destination nodes of all packets carried by them to decide which packets should be forwarded, which is a non-trivial burden for resource-limited MONs. Therefore, it is desirable to investigate efficient MON routing algorithms that can overcome both shortcomings. On the other side, packet routing among different landmarks in MONs can provide potential interesting applications. For example, data communication (i.e., Internet access) can be realized by relying on people or vehicles to carry and forward data when they move among rural villages and cities [35]. The concept of MONs has also been applied in animal tracking, which collects logged file from the digital collars attached to zebras in Kenya without infrastructure network [57]. However, few researches [36, 62, 65, 67, 73, 93] have been conducted to investigate the location information in node mobility to realize this function efficiently. Consequently, it is a challenging problem to realize efficient packet routing among different landmarks in MONs.

1.2 Research Approaches

Extensive investigations have been conducted to solve the two problems introduced in Section 1.1 in this dissertation. We briefly introduce our solutions in below.

1.2.1 Efficient File Sharing in MONs

1.2.1.1 Optimal File Replication for Efficient File Sharing in MONs

In this method, we consider the basic scenario, in which each node can only access the requested file when meeting the file holder. Following this setting, we first introduce a new concept of resource for file replication, which considers both node storage and node meeting ability. We then theoretically study the influence of resource allocation on the average file access delay. Based on the analysis result, we derive an optimal file replication rule that allocates resources to each file based on its popularity and size. We further propose a file replication protocol based on the rule, which approximates the minimum global file access delay in a fully distributed manner. The detail of this work will be introduced in Chapter 3.

1.2.1.2 Leveraging Social Networks for Efficient File Sharing in MONs

We then expand the investigation to the scenario in which nodes can actively forward file requests to reach the file holders. The requested file can also be forwarded among nodes to reach the requester. In this case, by leveraging nodes' social network properties on interest and movement pattern, we propose a Social network based Peer-to-peer (P2P) cOntent-based file sharing system in mobile Opportunistic Networks (SPOON). SPOON classifies common-interest and frequently-encountered nodes into social communities. Then, SPOON considers the frequency at which a node meets different interests rather than different nodes in file searching. SPOON also chooses stable nodes in a community as coordinators and highly mobile nodes that travel frequently to foreign communities as ambassadors. Such a structure ensures that a request can be forwarded to the community of the requested file quickly. SPOON also incorporates additional strategies for file prefetching, querying-completion and loop-prevention, and node churn consideration to further enhance file searching efficiency. The detail of this work will be introduced later in Chapter 4.

1.2.2 Efficient Packet Routing in MONs

1.2.2.1 Efficient Packet Routing among Nodes in MONs

In order to overcome the shortcomings of current routing algorithms in MONs, we propose SMART, which utilizes distributed social map for lightweight packet routing among nodes in mobile opportunistic networks. The design of SMART is inspired by the social network property that the people a person frequently meets are usually stable. These people also play an important role in forwarding packets for the person [17]. For example, we often meet the same colleagues, friends, and family members daily, and we often rely on them to forward messages to others. Consequently, SMART mainly applies to scenarios in which node carriers belong to certain social structures.

Specifically, in SMART, each node builds a *social map* to record its surrounding social network in MONs, which is constructed by learning each encountered node's most frequently met nodes (i.e., stable friends). Each link in the social map is associated with a weight based on the encountering frequency and social closeness of the two connected nodes. The weight is used to deduce the delivery abilities among nodes. Then, a node can decide whether to forward a packet to an encountered node by only checking its own social map, which can save the cost. Further, the social map is not limited to one or two hops and reflects possible long relay paths to provide better forwarder selection. As a result, packets can be routed to their destinations efficiently with a low cost. The detail of this work will be introduced later in Chapter 5.

1.2.2.2 Efficient Packet Routing among Landmarks in MONs

In current algorithms, packets are forwarded to gradually through nodes with higher probability of visiting the destination node or area. However, the number of such nodes usually is limited, leading to insufficient throughput performance. Therefore, in order to realize efficient packet routing among different landmarks in MONs, we propose an inter-landmark packet routing algorithm, called DTN-FLOW, that fully utilizes all node movements in MONs. DTN-FLOW selects popular places that nodes visit frequently as landmarks and divides the entire DTN area into sub-areas represented by landmarks. Nodes transiting between landmarks relay packets among landmarks, even though they rarely visit the destinations of these packets. Specifically, the frequency of node transition between two landmarks is measured to represent the forwarding capacity between them, based on which routing tables are built on each landmark to guide packet routing. Each node predicts its

transits based on its previous landmark visiting records using the order-k Markov predictor. When routing a packet, the landmark determines the next hop landmark based on its routing table, and forwards the packet to the node with the highest probability of transiting to the selected landmark. Thus, DTN-FLOW fully utilizes all node movements to route packets along landmark based paths to their destinations. The detail of this work will be introduced later later in Chapter 6.

1.3 Contributions

We summarize the contributions of this dissertation below.

- We investigate how to optimally create replicas in MONs to maximize file availability and consequently, improve the file sharing efficiency.
  (1) We introduce a new concept of available resource for file replication, which considers both storage and meeting ability. (ICNP'11 [19] and TC'14 [24])
  (2) We propose a globally optimal resource allocation rule that can lead to the minimum overall file access delay under the restriction of limited resources. (ICNP'11 [19] and TC'14 [24])
  (3) We design a novel file replication protocol to realize the proposed optimal file replication rule, which works in a fully distributed way and can be adapted to different MON scenarios. (ICNP'11 [19] and TC'14 [24])

- We leverage social network properties to enhance file sharing efficiency in MONs.
  (1) We propose a method to create interest based community for efficient file sharing in MONs. (MASS'11 [25] and TMC'12 [26])
  (2) We design novel intra-community and inter-community file searching algorithms based on the interest based community structure. (MASS'11 [25] and TMC'12 [26])

- We propose a social map based routing algorithm for MONs that can provide efficient packet routing among nodes with a low cost.
  (1) We propose a lightweight distributed social map construction algorithm to enable each node to build its social map and discover its surrounding social network. (ICNP'12 [20] and TON'13 [21])
  (2) We propose a new routing algorithm that has a high efficiency and a low cost based on the social map. (ICNP'12 [20] and TON'13 [21])

- We design a routing algorithms for MONs that can realize high throughput packet routing among different landmarks in the network.

  (1) We propose a landmark based sub-area division algorithm in MONs and model node mobility as the transit among landmarks. (IPDPS'13 [22] and TON'13 [23])

  (2) We measure the frequency of node transit between two landmarks as the packet forwarding capacity between them, based on which routing tables are built on each landmark to guide the packet forwarding. (IPDPS'13 [22] and TON'13 [23])

  (3) We further predict node mobility based on the order-k Markov predictor [86]. We then realize efficient file routing among landmarks based on aforementioned components. (IPDPS'13 [22] and TON'13 [23])

1.4 Dissertation Organization

The remainder of this thesis is arranged as followings. Chapter 2 presents the related work. Chapter 3 introduces the proposed method that can realize the optimal file replication for efficient file sharing in MONs. Chapter 4 presents the proposed method that utilizes social network properties for efficient file sharing in MONs. Chapter 5 introduces the proposed method that can realize efficient packet routing among nodes in MONs. Chapter 6 presents the proposed method that can realize efficient packet routing among landmarks in MONs. Finally, Chapter 7 concludes this dissertation with discussions on future work.

Chapter 2

Related Work

Mobile opportunistic networks have attracted significant interests due to its distinctive features and the increasing popularity of mobile devices. Extensive investigates have been conducted on the efficient file sharing and packet routing in the contexts of MONs and mobile ad hoc networks (MANETs). In this chapter, we summarize related works in this area. We first introduce previous methods on efficient file sharing in MANETs/MONs. We then present the-state-of-art algorithms for efficient packet routing in MONs that exploit encountering records, social network properties, and location information, respectively.

2.1 File Sharing in MANETs/MONs

2.1.1 File Sharing in MANETs

2.1.1.1 Efficient File Replication in MANETs

In [33, 45, 94], individual or a group of nodes decide the list of files to replicate according to file querying frequency. Hara [45] proposed three file replication protocols: Static Access Frequency (SAF), Dynamic Access Frequency and Neighborhood (DAFN), and Dynamic Connectivity based Grouping (DCG). In SAF, each node replicates its frequently queried files until its available storage is used up. SAF may lead to many duplicate replicas among neighboring nodes when they have the same interested files. DAFN eliminates duplicate replicas among neighbors. DCG further reduces duplicate replicas in a group of nodes with frequent connections by creating replicas for files in

the descending order of their group based querying frequencies. Though DAFN and DCG enable replicas to be shared among neighbors, neighboring nodes may separate from each other due to node mobility. Also, they incur high traffic load in identifying duplicates or managing groups.

Zhang *et al* [94] proposed to let each node collect data access statistics from neighbors to decide the creation or relinquishment of a replica. Duong and Demeure [33] proposed to group nodes with stable connections and let each node checks its group members' potential possibility of requesting a file and their storage status to decide replicate the file or not. Yin and Cao [91] proposed to cache popular files on the intersection nodes of file retrieval paths. Though it is effective for popular files, it fails to utilize all storage space on nodes.

Gianuzzi [41] investigated the probability of acquiring a file, which has n replicas in the network, from the potentially partitioned network. He also studied the file retrieval performance when erasure coding [28] is employed. Chen [27] discussed how to decide the minimal number of mobile servers needed to ensure that every data item can be obtained within at most k ($k \geq 1$) hops by any node in the system. Moussaoui *et al.* [76] proposed two steps of file replication, primary replication and dynamic replication, to disseminate replicas in the network evenly in order to meet user needs and prevent data loss in the case of network partition.

2.1.1.2 Flooding-based File Sharing Methods

In flooding-based methods, 7DS [80] is one of the first approaches to port P2P technology to mobile environments. It exploits the mobility of nodes within a geographic area to disseminate web content among neighbors. Passive Distributed Indexing (PDI) [71] is a general-purpose distributed file searching algorithm. It uses local broadcasting for content searching and sets up content indexes on nodes along the reply path to guide subsequent searching. Klemm *et al.* [61] proposed a special-purpose on-demand file searching and transferring algorithm based on an application layer overlay network. The algorithm transparently aggregates query results from other peers to eliminate redundant routing paths. Anna Hayes *et al.* [46] extended the Gnutella system to mobile environments and proposed the use of a set of keywords to represent user interests. However, these flooding-based methods produce high overhead due to broadcasting.

2.1.1.3 Advertisement-based File Sharing Methods

Tchakarov and Vaidya [88] proposed GCLP for efficient content discovery in location-aware ad hoc networks. It disseminates contents and requests in crossed directions to ensure their encountering. P2PSI [49] combines both advertisement (push) and discovery (pull) processes. It adopts the idea of swarm intelligence by regarding shared files as food sources and routing tables as pheromone. Each file holder regularly broadcasts an advertisement message to inform surrounding nodes about its files. The discovery process locates the desired file and also leaves pheromone to help subsequent search requests. Repantis and Kalogeraki [82] proposed a file sharing mechanism in which nodes use the Bloom filter to build content synopses of their data and adaptively disseminate them to other nodes to guide queries. Though the advertisement-based methods reduce the overhead of flooding-based methods, they still generate high overhead for advertising and cannot guarantee the success of file searching due to node mobility.

2.1.2 File Sharing in MONs

2.1.2.1 Cache/Replication-based File Sharing Methods

Huang *et al.* [51] proposed a method that considers multiple factors (e.g., node mobility, file popularity, and file server topology) in creating file replicas in file servers to realize optimal file availability on content distribution community. Gao *et al.* [39] proposed cooperative caching in disruption tolerant networks. It replicas each file to network central locations, which are frequently visited by nodes in the system, to ensure efficient data access. QCR [81] uses file caching to realize effective multimedia content dissemination in opportunistic networks. In addition to node mobility and file popularity, it also considers the impatience of users when creating replicas. Lenders *et al.* [66] investigated wireless ad hoc podcasting, in which nodes store contents from their neighbors that are interested by themselves or nodes they have met. Though these methods improve file availability, nodes in these methods passively wait for their interested contents rather than actively search files, which may lead to a long search delay.

2.1.2.2 Social Network-based File Sharing Methods

Social networks have also been utilized in content publishing/dissemination algorithms [11, 29,68,92] in opportunistic networks. MOPS [68] provides content-based sub/pub service by utilizing

the long-term neighboring relationship between nodes. It groups nodes with frequent contacts and selects nodes that connect different groups as brokers, which are responsible for inter-community communication. Then contents and subscriptions are relayed through brokers to reach different communities. MOPS only considers node mobility while SPOON is more advantageous by considering both node interest and mobility as described previously. Moreover, unlike MOPS that only depends on the meeting of brokers for inter-community search, SPOON enhances the efficiency of inter-community search by (1) assigning one ambassador for each known foreign community, which helps to forward a query directly to the destination community, and (2) utilizing stable nodes (coordinator) to receive messages from ambassadors.

The work in [92] is a similar to MOPS. It selects centrality nodes as brokers and builds them into an overlay, in which brokers use unicast or direct protocols (e.g., WiFi access points) for communication. Then node publications are first transferred to the broker node responsible for the node's community and then propagated to all brokers to find matched subscribers. SocialCast [29] calculates a node's utility value on an interest based on the node's mobility and co-location with the nodes subscribed to the interest. It publishes contents on an interest to subscribers by forwarding the contents to nodes with the highest utilities on the interest. ContentPlace [11] defines social relationship based communities and a set of content caching policies. Specifically, each node calculates a utility value of published data it has met based on the data's destination and its connected communities, and caches the data with the top highest utilities. The work in [38] considers social contact patterns and interests of mobile users to estimate users' potential interests in generated files and thereby realize efficient data dissemination.

2.2 Packet Routing in MONs

2.2.1 Probabilistic Routing Methods

Probabilistic routing methods [9, 14, 64, 72] use nodes' past encounter records to predict their future encounter probabilities, which is used to rank the suitability of a node to carry a packet. PROPHET [72] updates the encountering probability between two nodes when they meet and ages the probability over time. A packet is always forwarded to nodes with higher probability of meeting its destination. MaxProp [14], RAPID [9], and MaxContribution [64] extend PROPHET by further specifying forwarding and storing priorities based on the probability of successful delivery. Packets

with higher priorities are forwarded first, and high priority packets replace low priory packets when a node's storage is full.

2.2.2 Social Network-based Routing Methods

Considering that people carrying mobile devices usually belong to certain social relationships, social network based routing algorithms [10,29,30,53,68,92] exploits social network properties in DTNs for packet routing. MOPS [68] is a publish-subscribe system. It groups frequently encountered nodes into a cluster for efficient intra-community communication and selects nodes having frequent contacts with foreign communities for inter-community communication. BUBBLE [53] uses two layers of ranks: global and local. The global ranking is used to forward a packet to the destination community, and the local ranking helps to find the destination within the community. SimBet [30] adopts centrality and similarity to rank the suitability of a node to carry a packet. It is based on the concept that nodes having high centrality and similarity with the destination node tend to meet it frequently. The event dissemination system in [92] is similar to MOPS. It groups well-connected nodes into communities and selects nodes with the highest closeness centrality as brokers for inter-community dissemination. Costa *et al.* proposed a social network based publish-subscribe system [29]. It forwards messages to nodes that meet subscribers of the packet's interest category frequently and have high connectivity with other nodes. HiBop [10] defines node context by jointly considering various information, including personal interests, residence and work, and forwards packets to the nodes that have frequent encounter records with the context of the destination.

2.2.3 Location-based Routing Methods

Location based routing methods [36, 62, 65, 67, 73, 93] use previous geographical location to assist packet routing in DTNs. GeoDTN [73] encodes historical geographical movement information in a vector to predict the possibility of two nodes becoming neighbors. Then, packets are forwarded to nodes that are more likely to be a neighbor of the destination node. PGR [62] uses observed node mobility pattern to predict nodes future movement to forward packets to a certain geographical destination. GeoOpps [67] exploits the navigation system to calculate the minimal estimated time of delivery (METD) by considering the closest point of possible routes to the destination, and forwards

packets to vehicles that lead to smaller METD. In MobyPoints [65], a node's meeting probabilities with all possible locations are encoded in vectors. Then, forwarding decisions are made based on the similarity score between the vectors of relay node and destination node. In GeoComm [36], the geo-centrality of each geo-community is calculated based on its contact probabilities with each node. Such centralities are then exploited to realize efficient packet dissemination in DTNs. In PER [93], a node's past transits among landmarks or sojourn on landmarks are summarized to predict its probability of visiting a landmark within a time limit. Such information is further exploited to deduce two node's future contact probability for packet routing. LOUVRE [63] builds landmarks on road intersections and uses the landmark overlay for routing in vehicle networks. However, LOUVRE focuses on vehicular networks in which GPS and map are used to determine the connected landmark and the next landmark the node is moving toward.

Chapter 3

Optimal File Replication for Efficient Sharing in MONs

In this chapter, considering mobile nodes can only communicate with each other when they meet, i.e., within the communication of each other, in MONs, we consider the scenario in which nodes rely on the encountering with the file holder to access interested file. In this scenario, file replication is an efficient way to enhance the file availability for efficient file sharing. On the other hand, a node's frequency to meet other nodes (meeting ability in short) also influences the availability of its files. Files in a node with higher meeting ability have higher availability. Therefore, we introduce a new concept of resource for file replication, which considers both node storage and node meeting ability. We then theoretically study how to assign limited resources on nodes for the creation of file replication so that the global file access delay is minimized. We consider both connected MONs and disconnected MONs in the analysis. The former has a relatively dense node distribution in an area while the latter has sparsely distributed nodes that meet each other intermittently. We develop an optimal file replication rule (OFRR) in MONs following the analysis result. We also propose a novel distributed file replication protocol, denoted by PCS, to realize the replication rule.

In the following, we first present the modeling of the influence of the resource allocation on file availability under two representative mobility models in Section 3.1 and then introduce the design of the PCS file replication protocol in Section 3.2. Sections 3.3 and 3.4 present the performance evaluation of the proposed PCS algorithm in connected and disconnected MONs, respectively.

Finally, Section 3.5 summarizes this chapter.

3.1 Theoretical Analysis of Globally Optimal File Replication

3.1.1 Node Movement Models

We consider two types of MONs (i.e., connected MONs and disconnected MONs) for theoretical analysis. The former has a dense node distribution and is similar to the mobile ad hoc networks (MANETs) [7]. Since the random waypoint model (RWP) [13] is often used for the research on MANETs [41, 45, 94], we also use it to represent node mobility in connected MONs. The latter has sparse node distribution and therefore is a typical form of delay tolerant networks (DTNs) [56]. Therefore, we adopt the the community-based mobility model [77] to represent node mobility in disconnected MONs, which is often used in the research on DTNs [26, 29, 78]. We leave the analysis for other mobility models, e.g., those created by Bonn Motion Tool [1], to future work.

3.1.1.1 Random Waypoint Model for Connected MONs

We use the random waypoint model (RWP) [13] to model node mobility in connected MONs. In RWP, nodes repeatedly move to a randomly selected point at a random speed, which means each node has roughly similar probability to meet other nodes. However, nodes usually have different probabilities of meeting nodes in reality, i.e., nodes with faster speed can meet others more frequently. We hence let each node have a randomly obtained speed, rather than a continuously varying speed as in the normal RWP model.

3.1.1.2 Community-based Mobility Model for disconnected MONs

We use the community-based mobility model [77] to model node mobility in disconnected MONs. In this model, the entire test area is split into different sub-areas, denoted by caves. Each cave holds one community. A node belongs to one or more communities (i.e., home community). The routines and (or) social relationships of a node tend to decide its mobility pattern. While moving,

a node has probability P_{in} to stay in the home community and probability $1 - P_{in}$ to visit a foreign community. A node moves within its home communities for most of the time (i.e., P_{in} usually is large). Please refer to [77] for more details of this mobility model.

3.1.1.3 Assumptions and Limitations

With above two mobility models, our analysis relies on two assumptions: 1) the probability of meeting a certain node is the same for all nodes (RWP model) or all nodes in its home community (community-based model) and 2) nodes move independently in the network (both models). The two assumptions may not always hold in real cases, which limits the applicability of the analysis results. However, the analysis results can provide instructions on file replication because the two models can represent key characteristics in real scenarios and have been widely used in research works [26, 29, 41, 45, 78, 94]. We briefly discuss how to expand the analysis to general scenarios without the two assumptions in Section 3.1.3.3.

3.1.2 Modeling the Replication Optimization Problem

We present the general process to model the expected file access delay with file replication. Note that in this research, we only consider the scenario in which a file requester relies on the encountering with the file holder to obtain the file. We let $\hat{m}_i$ be the probability that a node's newly met node in the coming time interval is node i, which reflects the meeting ability of files on node i. We also use X_{ij} to denote whether node i owns file j (or its replica). Then, the average number of time intervals needed to meet a specific file, say file j, can be represented as:

$$\overline{\hat{T}_j} = \frac{1}{\sum_{i=1}^{N} \hat{m}_i X_{ij}} \tag{3.1}$$

Then, the average number of intervals needed to satisfy a request is

$$\overline{\hat{T}} = \sum_{j=1}^{F} q_j \overline{\hat{T}_j} = \sum_{j=1}^{F} \frac{q_j}{\sum_{i=1}^{N} \hat{m}_i X_{ij}}, \tag{3.2}$$

where q_j is the probability of requesting file j. With Formula (3.2), we can formulate the global optimization problem as minimizing $\overline{\hat{T}}$, which can be utilized to deduce the optimal replication rule.

However, the calculation of $\hat{m}_i$ may be complex and makes the minimization problem non-trivial. We will discuss how this is handled with the two common mobility models in Section 3.1.3.

3.1.3 Theoretical Analysis

In this section, we theoretically analyze the influence of the file replica distribution on the

Table 3.1: Notations in analysis.

Notation	Meaning
q_j	The prob. of requesting file j in the system
m_i	The prob. that the next encountered node is node i
p_j	The prob. of obtaining file j in the next encountered node
N	Total number of nodes
V_i	Node i's meeting ability (i.e., frequency of meeting nodes)
S_i	Storage space of node i
$\overline{V}$	Average meeting ability of all nodes in the system
F	Total number of files in the system
b_j	Size of file j
X_{ij}	Whether node i contains file j or not
V_{jk}	Meeting ability of the k^{th} node that holds file j
n_j	The number of nodes holding file j or its replicas
A_j	Allocated resource for file j for replication
$\overline{T_j}$	Average number of time intervals needed to meet file j
$\overline{T}$	Average number of time intervals needed to meet a file
$\mathcal{R}$	Total amount of resource in the system
P_j	Priority value of file j, $P_j = \sqrt{q_j/b_j}$

overall file access delay in MONs under the two mobility models following the process introduced in Section 3.1.2. Please refer to Table 3.1 for the notations in our analysis.

3.1.3.1 Optimal File Replication with the RWP model

In the RWP model, we can assume that the inter-meeting time among nodes follows exponential distribution [15,43]. Then, the probability of meeting a node is independent with the previous encountered node. Therefore, we define the **meeting ability** of a node as the average number of nodes it meets in a unit time and use it to investigate the optimal file replication. Specifically, if a node is able to meet more nodes, it has higher probability of being encountered by other nodes later on. We use m_i to denote the probability that the next node a file request holder meets is node i. Then, m_i is proportional to node i's meeting ability (i.e., V_i). That is

$$m_i = \frac{V_i}{\sum_{k=1}^{N} V_k} = \frac{V_i}{\overline{V}N} \tag{3.3}$$

where N denotes the total number of nodes and $\overline{V}$ denotes the average meeting ability of all nodes.

We use vector $(V_{j1}, V_{j2}, \ldots, V_{jn_j})$ to denote the meeting abilities of a group of nodes holding file j or its replica, where n_j is the number of file j (including replicas) in the system. Then, the probability that a node obtains its requested file j from its encountering node is the sum of the

probabilities of encountering nodes that hold file j or its replica. That is,

$$p_j = \sum_{i=1}^{N} m_i X_{ij} = \sum_{i=1}^{N} \frac{V_i}{\overline{V}N} X_{ij} = \sum_{k=1}^{n_j} \frac{V_{jk}}{\overline{V}N} \tag{3.4}$$

where X_{ij} is a zero-one variable that denotes whether node i contains file j (or its replica).

As stated above, a node's probability of being encountered by other nodes is proportional to its meeting ability. This indicates that files residing in nodes with higher meeting ability have higher availability than files in nodes with lower meeting ability. So we take into account both meeting ability and storage in measuring a node's resource. When a replica is created on a node, it occupies the memory on the node. Also, its probability of being met by others is decided by the node's meeting ability. This means that the replica naturally consumes both the storage resource and the meeting ability resource of the node. Therefore, we denote the resource on a node by $S_i V_i$, in which S_i denotes node i's storage space and V_i denotes its meeting ability. Then, the total amount of resources in the system ($\mathcal{R}$) is:

$$\mathcal{R} = \sum_{i=1}^{N} S_i V_i \tag{3.5}$$

Thus, the total resources allocated to file j is:

$$R_j = b_j \sum_{k=1}^{n_j} V_{jk} \tag{3.6}$$

where b_j is the size of file j. Based on Equation (3.6), Equation (3.4) can be represented as

$$p_j = \frac{b_j \sum_{k=1}^{n_j} V_{jk}}{b_j \overline{V} N} = \frac{R_j}{b_j \overline{V} N} \tag{3.7}$$

Thus, the probability of meeting file j after k ($k = 1, 2, 3, \cdots$) time intervals (i.e., average inter-meeting time among nodes) is

$$(1 - p_j)^{k-1} p_j$$

and the average number of time intervals needed for a node to meet a node containing file j is

$$\overline{T_j} = \sum_{k=1}^{\infty} k(1 - p_j)^{k-1} p_j = \frac{1}{p_j} = \frac{b_j \overline{V} N}{R_j} \tag{3.8}$$

We use $q_j \in [0, 1]$ to denote the probability of generating a request for file j in the system. Then, the average number of intervals needed to satisfy a request is

$$\overline{T} = \sum_{j=1}^{F} q_j \overline{T_j} = \sum_{j=1}^{F} q_j \frac{b_j \overline{V} N}{R_j} = \overline{V} N \sum_{j=1}^{F} \frac{q_j b_j}{R_j} \tag{3.9}$$

We aim to minimize the global file access delay (i.e., $\overline{T}$) by file replication. According to Equation (3.9), $\overline{T}$ is decided by q_j, b_j and R_j, and the values of q_j and b_j are decided by the system. Thus, the problem of optimal resource allocation is then converted to finding the optimal amount of resources (R_j) for each file j under the restriction of total available resources in order to achieve the minimum average access delay.

Suppose $B_j = q_j b_j$, with Equations (3.5) and (3.9), the problem of optimal resource allocation is expressed by

$$\min(\overline{T}) = \min\{\sum_{j=1}^{F} \frac{q_j b_j}{R_j}\} = \min\{\sum_{j=1}^{F} \frac{B_j}{R_j}\} \tag{3.10}$$

subject to:

$$\sum_{j=1}^{F} R_j \leq \mathcal{R}.$$

Equation (3.9) also indicates that each R_j should be as large as possible in order to minimize $\overline{T}$. Therefore, we assume all resources ($\mathcal{R}$) are allocated.

$$\sum_{j=1}^{F} R_j = \mathcal{R} \tag{3.11}$$

By applying Formula (3.11), Formula (3.10) is changed to

$$\min(\overline{T}) = \min\{\frac{B_1}{R_1} + \frac{B_2}{R_2} + \cdots + \frac{B_F}{\mathcal{R} - (R_1 + R_2 + \cdots + R_{F-1})}\} \tag{3.12}$$

Next, we try to find the value of R_j ($1 \leq j \leq F-1$) that satisfies Formula (3.12). Specifically, we first calculate the first order (necessary) condition by differentiating $\overline{T}$ on each R_j ($1 \leq j \leq F-1$), respectively, and find the value of R_j that makes the differentiated formula equal 0. The resultant formulas after differentiation are

$$\frac{B_1}{R_1^2} - \frac{B_F}{\{\mathcal{R} - (R_1 + R_2 + \cdots + R_{F-1})\}^2} = 0 \tag{3.13}$$

$$\cdots\cdots\cdots$$

$$\frac{B_{F-1}}{R_{F-1}^2} - \frac{B_F}{\{\mathcal{R} - (R_1 + R_2 + \cdots + R_{F-1})\}^2} = 0 \tag{3.14}$$

Combine all above $F-1$ equations, we get

$$\frac{B_1}{R_1^2} = \frac{B_2}{R_2^2} = \frac{B_3}{R_3^2} = \cdots = \frac{B_{F-1}}{R_{F-1}^2} = \frac{B_F}{R_F^2} \tag{3.15}$$

To achieve the minimal average delay, the second order (sufficient) condition should be larger than 0. That is:

$$\frac{-2B_1}{R_1^3} - \frac{-2B_F}{\{\mathcal{R} - (R_1 + R_2 + \cdots + R_{F-1})\}^3} > 0 \tag{3.16}$$

. . . . . .

$$\frac{-2B_{F-1}}{R_{F-1}^3} - \frac{-2B_F}{\{\mathcal{R} - (R_1 + R_2 + \cdots + R_{F-1})\}^3} > 0 \tag{3.17}$$

If Equation (3.15) is true, based on Equation (3.11), Formulas (3.16) and (3.17) can be transformed to below.

$$(\frac{1}{R_F} - \frac{1}{R_1})\frac{2B_1}{R_1^2} > 0 \tag{3.18}$$

. . . . . .

$$(\frac{1}{R_F} - \frac{1}{R_{F-1}})\frac{2B_{F-1}}{R_{F-1}^2} > 0 \tag{3.19}$$

When $R_F < R_j$ ($j \in [1, F-1]$), Equations (3.18) and (3.19) (and also the second order condition) are satisfied. Recall that the above result is obtained when we replace R_F with $\mathcal{R}-(R_1+R_2+\cdots+R_{F-1})$ in Equation (3.10). If we replace R_k ($k \in [1, F]$) with $\mathcal{R} - (R_1 + \cdots R_{k-1} + R_{k+1} + \cdots R_F)$, the second order is also satisfied when $R_k < R_j$ ($j \in [1, F], j \neq k$). In summary, the second order is satisfied when the resource allocated for one file is less than the resource allocated for any other file. This condition is always true because there always exists a file with the minimum allocated resource. Therefore, as long as the first order condition (i.e., Equation (3.15)) is satisfied, the second order condition is also satisfied.

Then, according to Equation (3.11) and Equation (3.15), we can see that the optimal allocation is

$$R_j = \frac{\sqrt{B_j}}{\sum_{k=1}^{F} \sqrt{B_k}} \mathcal{R} \ (j = 1, 2, 3, \cdots, F) \tag{3.20}$$

This means that the optimal resource allocation is achieved through the square root policy, i.e., the portion of resource for file j is in direct proportion of the square root of B_j:

$$R_j \propto \sqrt{B_j} \Rightarrow b_j \sum_{k=1}^{n_j} V_{jk} \propto \sqrt{b_j q_j} \tag{3.21}$$

That is

$$\sum_{k=1}^{n_j} V_{jk} \propto \sqrt{\frac{q_j}{b_j}} \Rightarrow \sum_{k=1}^{n_j} V_{jk} \propto P_j \tag{3.22}$$

We call $\sqrt{q_j/b_j}$ the *Priority Value* (P) of file j as it represents the relative priority in acquiring resource for the global optimization on file access delay.

Based on Formula (3.22), we derive the Optimal File Replication Rule (OFRR) that gives the direction for the optimal resource allocation that leads to the minimum average file access delay

under the RWP model.

OFRR. *In order to achieve the minimum overall file access delay, the sum of the meeting abilities of replica nodes of file j should be proportional to $P_j = \sqrt{q_j/b_j}$.*

3.1.3.2 Optimal File Replication with the Community-based Mobility Model

In this section, we conduct the analysis under the community-based mobility model. Unless otherwise specified, we use the same notations in Table 3.1 (which is for the RWP model) but add $'$ to each notation to denote that it is for the community-based mobility model. Recall that in the RWP model, we can assume that the inter-meeting time among nodes follows exponential distribution. Based on this assumption, we can calculate the probability that a newly met node is node i (i.e., m_i), which is used to find the expected time $\overline{T}$ to satisfy a request and finally deduce the OFRR that can minimize $\overline{T}$. However, under the community-based mobility model, this assumption does not hold [85]. This makes it difficult to calculate m_i, which makes the process of minimizing the overall delay $\overline{T'}$ a formidable problem. To deal with this problem, rather than considering meeting ability, we consider each node's **satisfying ability**. It is defined as a node's ability to satisfy requests in the system (denoted by V_i') and is calculated based on its capacity to satisfy requests in each community.

We use N_c to denote the number of nodes in community c. Then, community c holds $\frac{N_c}{N}$ fraction of nodes in the system. Node i's satisfying ability to community c depends on both the number of different nodes in c it meets in a unit time period (denoted by M_{ic}) and the number of requests generated by nodes in c. In this model, since nodes' file interests are stable during a certain time period, we assume that each node's file requesting pattern (i.e., requesting rates for different files) remains stable in the considered period of time.

Then, the number of nodes in a community represents the number of requests for a given file generated in this community. As a result, a file holder has low ability to satisfy requests from a small community. Thus, we integrate each community's fraction of nodes (i.e., $\frac{N_c}{N}$) into the calculation of the satisfying ability. Therefore,

$$V_i' = \sum_{c=1}^{C} M_{ic} \frac{N_c}{N} \tag{3.23}$$

where C is the total number of communities.

Given n_j nodes that hold file j or its replica, we again use vector $(V'_{j1}, V'_{j2}, \ldots, V'_{jk}, \ldots, V'_{in_j})$ to denote the satisfying abilities of these nodes. Then, the overall ability of nodes in the system to

satisfy requests for file j (denoted by O_j) is the sum of all the satisfying abilities times a redundancy elimination factor α.

$$O_j = \alpha \sum_{k=1}^{n_j} V'_{jk} \ (\alpha \in [0,1]) \tag{3.24}$$

α is added because different holders of file j may meet the same requester for file j in the same time unit. Since the requester has only one request for file j, only the first meeting satisfies the file request, and the subsequent meetings do not satisfy a request for file j. In other words, α denotes the "discount" on the overall satisfying ability considering the fact that the satisfying abilities of different file holders may overlap.

Then, the number of time intervals (i.e., average inter-meeting time among nodes) needed to satisfy a request for file j is

$$\overline{T'_j} = \frac{1}{O_j} = \frac{1}{\alpha \sum\limits_{k=1}^{n_j} V'_{jk}} \tag{3.25}$$

Recall that b_j denotes the size of file j and q_j denotes the probability of initiating a request for file j from nodes in the system. Similar to Equation (3.6), the total resources (satisfying resource and storage resource) allocated to file j can be represented by $R'_j = b_j \sum\limits_{k=1}^{n_j} V'_{jk}$. As a result, the average number of time intervals needed to satisfy a request in the system is

$$\overline{T'} = \sum_{j=1}^{F} q_j \overline{T'_j} = \sum_{j=1}^{F} q_j \frac{1}{\alpha \sum\limits_{k=1}^{n_j} V'_{jk}} = \frac{1}{\alpha} \sum_{j=1}^{F} \frac{q_j b_j}{R'_j} \tag{3.26}$$

Then, the problem of optimal resource allocation can be expressed by

$$\min(\overline{T'}) = \min\{\sum_{j=1}^{F} \frac{q_j b_j}{R'_j}\} = \min\{\sum_{j=1}^{F} \frac{B_j}{R'_j}\} \tag{3.27}$$

subject to:

$$\sum_{j=1}^{F} R'_j \leq \mathcal{R}.$$

We can find that Equation (3.27) is the same as Equation (3.10). Then, we follow the same process after Equation (3.10) and deduce the OFRR rule in disconnected MONs as

$$\sum_{k=1}^{n_j} V'_{jk} \propto \sqrt{\frac{q_j}{b_j}} \Rightarrow \sum_{k=1}^{n_j} V'_{jk} \propto P_j \tag{3.28}$$

We see that the OFRR under the community-based mobility model (Equation (3.28)) is the same as the OFRR deduced with the RWP model (Equation (3.22)) except that V'_{jk} is the satisfying ability (Equation (3.23)) in the former while is the meeting ability (defined in Table 3.1) in the later. It is intriguing to find that Equation (3.23) turns to be the same as the definition of V_i in Table 3.1

if the number of community is 1. This means that the OFRR expressed by Equation (3.22) is a special case of the OFRR expressed by Equation (3.28). **As a result, our previously deduced OFRR can be the OFRR for MONs under the two mobility models.**

It is interesting to find that the OFRR follows the "square root assignment rule" derived by Kleinrock [60] for the link capacity assignment in wireless communication to maximize the network efficiency. It also matches with the findings in [58] that when file servers may be unavailable due to node dynamism, the wired P2P content distribution system can achieve the maximum file hit rate when available storage is allocated in proportion to a constant value plus $ln(q_j/b_j)$ for each file.

3.1.3.3 Extension to General Node Mobility Models

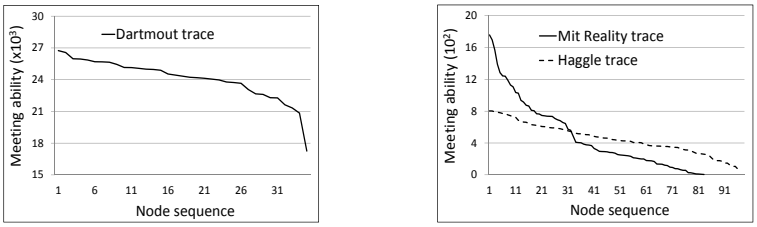
In above two subsections, we deduced the OFRR rule in RWP mobility model and community-based mobility model following the basic idea in Section 3.1.2. However, above analysis relies on two assumptions mentioned in Section 3.1.1.3, which may not hold in general node mobility models. Therefore, it is non-trivial to extend above analysis to general cases directly. Specifically, in certain mobility models, different nodes may have different visiting preferences or patterns, making different node's probabilities of meeting node i in the next encountering ($\hat{m_i}$) lack a general expression.

However, there are some ways to make the analysis in general cases possible. For example, we can incorporate new factors into $\hat{m_i}$ to express each node's distinct patterns, e.g., active levels and community identities. These factors usually represent how frequent a node meets others. We can also first measure the meeting abilities of different nodes in a real scenario and then assign labels to each node to indicate its rough meeting ability. With these simplifications, $\hat{m_i}$ can be expressed and the analysis can be conducted.

On the other hand, there are possibly fixed nodes in the system, which are naturally supported in our analysis. This is because we only care a node's storage and meeting ability regarding creating file replicas. Though fixed nodes do not move, they can meet other nodes, which means their meeting abilities can be measured or even formulated. As a result, fixed nodes are regarded the same as mobile nodes in the system.

3.1.4 Meeting Ability Distribution in Real Traces

We measured the meeting ability distribution from real traces to confirm the necessity of considering node meeting ability as an important factor in the resource allocation in our design.



(a) In a connected MON. (b) In disconnected MONs.

Figure 3.1: Meeting ability distribution.

Specifically, for connected MONs, we used the Dartmouth trace [42], which was obtained through an outdoor project in Dartmouth College. The trace provides position records of 35 laptop nodes moving randomly and independently across different sections of an open field. For disconnected MONs, we used the MIT Reality trace [34] and the Haggle trace [18]. In the former, 97 smart phones were distributed to students and faculties at MIT. In the latter, 98 iMotes were assigned to scholars attending the Infocom'06 conference. In both traces, the encountering among participating nodes was recorded.

For each trace, we measured the meeting abilities of all nodes and ranked them in decreasing order, as shown in Figure 3.1(a) and Figure 3.1(b). We see that in all three traces, node meeting ability is distributed in a wide range. This matches with our previous claim that nodes usually have different meeting abilities. Also, it verifies the necessity of considering node meeting ability as a resource in file replication. This is because if all nodes have similar meeting ability, replicas on different nodes have similar probability to meet requesters, and hence there is no need to consider meeting ability in resource allocation.

3.2 Distributed File Replication Protocol

In this section, we propose a distributed file replication protocol that can approximately realize the optimal file replication rule (OFRR) with the two mobility models in a distributed manner. Since the OFRRs in the two mobility models (i.e., Equation (3.22) and Equation (3.28)) have the same form, we present the protocol in this section without indicating the specific mobility model. We first introduce the challenges to realize the OFRR and our solutions. We then propose a replication protocol to realize OFRR distributively and analyze the effeciveness of the protocol.

3.2.1 Challenges and Solutions to Achieve the OFRR

Challenge 1: resource allocation without a central server. OFRR shows that in order to realize the globally optimal file access delay, each file's popularity (q_j) and size (b_j), and the system resource ($\mathcal{R}$) information (both node storage size and moving ability) must be known in order to decide the portion of resource for each file for replica creation. Specifically, suppose there are F files in the system with $b_1q_1 \cdots b_Fq_F$ and total resource $\mathcal{R}$, the resource allocated to file j (R_j) should be

$$R_j = \mathcal{R} \times \sqrt{b_jq_j} / \sum_{k=1}^{F} \sqrt{b_kq_k} \tag{3.29}$$

Then, an intuitive way to achieve this goal is to setup a central server to collect above information, conduct the resource allocation for each file, and distribute the information to file owners to replicate their files. However, the nature of the distributed network, node mobility, and transmission range constraint make the building of such a central server infeasible. For example, since nodes are constantly moving and have limited communication ranges, it is impossible for each node to update its information to or receive information from the server timely. Thus, a severe challenge is to enable a node to distributively figure out the proper portion of resource for each of its files without a central server.

Even when each node knows $\sqrt{b_jq_j} / \sum_{k=1}^{F} \sqrt{b_kq_k}$ of each of its files, the total amount of resources available in the system may change due to node joins and departures, which makes it difficult for a node to calculate the portion of resource for each of its file (R_j). For example, suppose there are only two files in the system, say f_1 and f_2, and the ratio of their allocated resources should be 4:1. If the total amount of resource $\mathcal{R} = 40$, the amount of resource allocated to f_1 is 32. If $\mathcal{R} = 60$, the amount for f_1 should be adjusted to 48. Further, the time-varying file popularity (q_j) makes the problem even more formidable. Therefore, OFRR cannot be simply realized by letting each node distribute replicas of a file until an absolute amount of resources are occupied.

Solution to Challenge 1: resource competition. OFRR (i.e, Formula (3.22)) requires that for each file, the sum of its replica nodes' meeting abilities, $\sum_{k=1}^{n_F} V_{Fk}$, is proportional to its priority value P. In other words, OFRR can be shown by

$$P_1 / \sum_{k=1}^{n_1} V_{1k} = P_2 / \sum_{k=1}^{n_2} V_{2k} \cdots = P_F / \sum_{k=1}^{n_F} V_{Fk} \tag{3.30}$$

where n_j ($j \in [1, 2, \cdots, F]$) represents the number of replica nodes of file j. Then, we can let each file, say file j, periodically compete for the resource with its current $P_j / \sum_{k=1}^{n_j} V_{jk}$. In one competition,

the file with the highest $P_j / \sum_{k=1}^{n_j} V_{jk}$ has the highest probability to win, i.e., creating one replica. After a file creates a replica, its $P_j / \sum_{k=1}^{n_j} V_{jk}$ decreases. The competition stops when all available resource is allocated. Thus, files with larger $P_j / \sum_{k=1}^{n_j} V_{jk}$ win more competitions and receive more resources and files with smaller $P_j / \sum_{k=1}^{n_j} V_{jk}$ only win few competitions and receive less resources. The competition gradually lets each file receive its deserved portion of resources based on OFRR. By enabling file owners to distributively compete for resource for their files, we can realize OFRR without a central server.

Challenge 2: competition for distributed resource. In a MON, available resources are scattered among different nodes moving around in the network. This poses three problems. First, different file owners are scattered and can hardly gather together to conduct the resource competition. Second, after a file is replicated to a number of nodes, it is difficult to collect the popularity of the replicas to update the P of the file. Third, since the number of nodes met by a file owner is limited, a single file owner cannot distribute replicas efficiently and quickly. We propose a work-around for this problem. Specifically, we regard a file and its newly created replica as two different files, which participate in further competition independently with evenly split priority value, i.e., $P/2$. However, this brings another challenge: since replica nodes of a file are scattered in the network, how to ensure that the overall $\sum V_{jk}$ is proportional to the overall P of the file?

Solution to Challenge 2: distributive competition on selective resources. In the solution to Challenge 1, each file periodically competes for resource with its current $P_j / \sum_{k=1}^{n_j} V_{jk}$. However, as previously mentioned, it is a challenge to keep the overall P proportional to the overall $\sum V_{jk}$ while replica holders are scattered. We indirectly resolve this problem by keeping the average V of the replica nodes of a file close to $\overline{V}$. Then, Formula (3.22) can be re-expressed as

$$n_j * \overline{V} \propto \sqrt{\frac{q_j}{b_j}} \Rightarrow n_j \propto \sqrt{\frac{q_j}{b_j}} \Rightarrow n_j \propto P_j \tag{3.31}$$

In such a case, when the number of replicas of each file is proportional to its $P_j = \sqrt{q_j / b_j}$, OFRR is satisfied. To attain this goal, we let each node deliberately select a neighbor node to create replicas for each file so that the average meeting ability of the file's replica nodes equals or is closest to $\overline{V}$. Considering the diverse node mobility in the network, a node can easily find replica nodes that satisfy the above requirement during its movement. Then, based on Equation (3.31), each node only needs to consider the P of each file in the resource competition. Upon winning a competition for a file,

a node splits the file's P evenly between the file and the replica. After this, the popularity of each file/replica is independently and continuously updated based on the number of requests received for it in a unit time period.

When a replica is deleted in the competition, we cannot reverse the process of priority split because it is very difficult to track the holders of the original file in a distributed manner due to node mobility in MONs. Fortunately, we can use the requesting popularity q to handle this problem. In this case, the qs (or Ps) of other replicas of the file increase since they can receive more requests for the file as the total amount of requests for the file is fixed, i.e., decided by the overall file popularity. That is, the sum of the replicas' Ps equals the overall P of the original file j (P_i). The increase of the priority values of other replicas of the file can be regarded as a reverse of the replica deletion. As a result, the number of replicas of each file is proportional to the sum of the meeting abilities of its replica nodes, thereby realizing Formula (3.22).

3.2.2 Design of the File Replication Protocol

The two solutions to handle the challenges in achieving the OFRR described above represent a maximal approximation to realize the OFRR in a distributed manner. Based on the solutions, we propose the Priority Competition and Split file replication protocol (PCS). We first introduce how a node retrieves the parameters needed in PCS and then present the detail of PCS.

In PCS, each node dynamically updates its meeting ability (V_i) and the average meeting ability of all nodes in the system ($\overline{V}$). Such information is exchanged among neighbor nodes. We explain the detail of this step in Section 3.2.3. Each node also periodically calculates the $P_j = \sqrt{q_j/b_j}$ of each of its files. The q_j is calculated by $q_j = u_j/U$, where u_j and U are the number of received requests for the file and the total number of requests generated in a unit of time period, respectively. Note that U is a pre-defined system parameter.

In the solution to Challenge 2, nodes replicate files distributively and select replicate nodes to ensure that the average meeting ability of replica nodes of a file the closest to $\overline{V}$. That is, $\overline{V_{n'_j}} \approx \overline{V}$, where $\overline{V_{n'_j}}$ is the average meeting ability of the replica nodes of file j and n'_j is the number of replica nodes of file j. Therefore, each node tracks n'_j and $\overline{V_{n'_j}}$ for each of its file. After creating a replica, the node increases n'_j by 1 and updates $\overline{V_{n'_j}}$ using the V of the new replica node.

With above information, we introduce the process of the replication of a file in PCS. Based on OFRR, since a file with a higher P should receive more resources, a node should assign higher

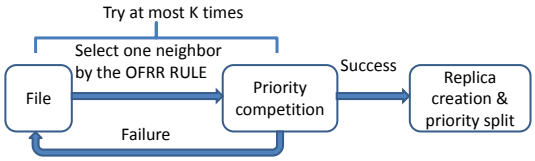


Figure 3.2: Replica distribution process.

priority to its files with higher P to compete for resource with other nodes. Thus, each node orders all of its files in descending order of their Ps and creates replicas for these files in a top-down manner periodically. Algorithm 1 presents the pseudo-code for the process of PCS between two encountered nodes. In detail, suppose node i needs to replicate file j, which is on the top of the list. Then, as shown in Figure 3.2, it keeps trying to replicate file j on nodes it encounters until one replica is created or K attempts have been made. If file j is replicated, its P is split. Next, the node fetches the file from the top of the list and repeats the process. If file j fails to be replicated after K attempts, the node stops launching competition until the next period.

Algorithm 1 Pseudo-code of PCS between node i and k.

```
i.createReplicasOn(k)     //node i tries to create a replica on node k
k.createReplicasOn(i)     //node k tries to create a replica on node i
Procedure createReplicasOn (node)
  nCount ← 0     //initialize a count
  this.orderFilesByP()     //order files by priority value
  For (each file f in current node)     //try to replica each file
      If (node.compete4File(f) == true)     //competition
          node.createAReplica4(f)     //create a replica if win
      else
           nCount ← nCount+1
      If nCount ≥ K     //try at most K times
          Break
end Procedure
Procedure compete4File()     //Compete for file j
  While (nRemainningMem < j.size())
      nSum ← nTotal ← nRandom ← fData ← 0     //initilization
      For (each file f (including j) in current node)
          nTotal ← nTotal+1/P_f
      nRandom ← generateARandomNumber() % nTotal
      For (each file f (including j) in current node)
          nSum ← nSum+1/P_f
          If (nSum >= nRandom)
            fData = f  Break     //pick the file
      If (fData = j)     //j is the picked file, competition fails
          return false
      Else     //win the competition
          select fData
  delSelectedFiles()     //delete the selected file
  return true
end Procedure
```

Following the solution to Challenge 2, a replicating node should keep the average meeting ability of the replica nodes for file j around $\overline{V}$. Node i first checks the meeting abilities of neighbors and then chooses the neighbor k that does not contain file j and makes $\overline{V_{n'_j}}^{new} = (n'_j\overline{V_{n'_j}}+V_k)/(n'_j+1)$ the closest to $\overline{V}$ as the replica node candidate. It is possible that $\overline{V_{n'_j}}^{new}$ is far away from $\overline{V}$. Therefore, we set a deviation range r. If creating a replica in the selected neighbor makes $(\overline{V_{n'_j}}^{new} - \overline{V}) > r$, the node does not replicate file j until it has a different set of neighbors.

In the case that $(\overline{V_{n'_j}}^{new} - \overline{V}) \leq r$, if the selected neighbor's available storage is larger than the size of file j (S_j), it creates a replica for file j directly. Otherwise, a competition is launched among the replica of file j and replicas already in the neighbor node based on their Ps. The priority value of the new replica is set to half of the original file's P. According to the solution to Challenge 1, the probability that a replica wins the resource competition is proportional to its P, i.e., a replica's probability of being selected to be removed is inversely proportional to its P. Then, suppose there are d replicas in competition, we let each replica be responsible for a range that equals its $1/P$ in space $[0, \sum_{k=1}^{d} 1/P_k]$. The neighbor node randomly chooses a number in $[0, \sum_{k=1}^{d} 1/P_k]$, and the replica whose range owns the number is selected to be removed. The neighbor node repeats above process until available storage is no less than the size of file j.

If file j is among the selected files, it fails the competition and will not be replicated in the neighbor node. Otherwise, all selected files are removed and file j is replicated. If file j fails, node i will launch another attempt for file j until the maximum number of attempts (K) is reached. The setting of K attempts is to ensure that each file can compete with a sufficient subset of replicas in the system. If node i fails to create a replica for file j after K attempts, then replicas in node i whose Ps are smaller than that of file j are unlikely to win a competition. Thus, at this moment, node i stops replicating files until next round. Finally, all available resources in the system are allocated to replicas according to their Ps, thereby realizing the OFRR.

According to the Solution to Challenge 2, we regard file j's replica as a "different" file in PCS. Therefore, if node i successfully creates a replica for file j, it splits the file's priority value P evenly between file j and the new replica. Thus, each file's priority is $P/2$. After the splitting, the two copies of file j involve in further resource competition independently. Note that we do not split file in the PCS algorithm but just split the priority value of a file when a replica of the file is created.

The replication for a file stops when the communication session of the two involved nodes ends. Then, each node continues the replication process for its files after excluding the disconnected

node from the neighbor node list. Since file popularity, Ps, and available system resources change as time goes on, each node periodically executes PCS to dynamically handle these time-varying factors. Each node also periodically calculates the popularity of its files (q_j) to reflect the changes on file popularity (due to the change of node requesting pattern) in different time periods. The periodical file popularity update can automatically handle file dynamism.

3.2.3 How to Collect Meeting Ability Information

To save communication cost, the values of V_i and $\overline{V}$ of each node are piggybacked into its beacon messages. Since V_i and $\overline{V}$ are only several bytes, the piggybacking only slightly increases the size of the beacon message. In connected MONs, a node's meeting ability (V_i) is simply measured as the frequency it meets other nodes. In disconnected MONs, a node needs to know the distribution of different communities to calculate its satisfying ability (Equation (3.23)). We then let each node piggyback its community ID and the community information it knows in the beacon message. We also let each node use the average meeting ability of all so far encountered nodes as that for all nodes in the system. As nodes meet more and more nodes, the calculated value can generally represent the average meeting ability of all nodes.

3.2.4 Analysis of the Effectiveness of PCS

In this section, we briefly prove the effectiveness of PCS. We refer to the process in which a node tries to copy a file to its neighbors as one round of replica distribution.

Recall that when a replica is created for a file with P, the two copies will replicate files with priority $P/2$ in the next round. This means that the creation of replicas will not increase the overall P of the file. Also, after each round, the priority value of each file or replica is updated based on the received requests for the file. Then, though some replicas may be deleted in the competition, the total amount of requests for the file remains stable, making the sum of the Ps of all replicas and the original file roughly equal to the overall priority value of the file. Then, we can regard the replicas of a file as an entity that competes for available resource in the system with accumulated priority P in each round. Therefore, in each round of replica distribution, based on our design of PCS, the overall probability of creating a replica for an original file j, denoted by Ps_j, is proportional to its

overall P_j. That is:

$$Ps_j \propto P_j \tag{3.32}$$

Then, suppose total M rounds of competition are conducted, the expected number of replicas, denoted by n_j, for file j is

$$n_j = MPs_j \Rightarrow n_j \propto P_j \tag{3.33}$$

Therefore, we conclude that the PCS algorithm can realize Equation (3.31), in which the number of replicas of each file is proportional to its P, thereby realizing the OFRR.

3.3 Performance Evaluation in Connected MONs based on the RWP Mobility Model

To evaluate the performance of PCS in connected MONs, we conducted experiments on both the GENI Orbit testbed [3, 4] and the NS-2 simulator [5]. The GENI testbed consists of 400 nodes equipped with wireless cards. We used the Dartmouth real-world trace [42], which provides the mobility trace of 35 laptops moving in an open field, to drive node mobility in both experiments. In order to validate the adaptability of PCS, we used two routing protocols in the experiments. We first used the StaticWait protocol [87] in the GENI experiment, in which each request stays on the source node waiting for the destination. We then used a probabilistic routing protocol (PROPHET [72]), in which a node routes requests to the neighbor with the highest meeting ability. We set a larger TTL for Static Wait since it needs more time to find a file holder. In addition to above tests, we also conducted simulation on the NS-2 with different network sizes and node mobility synthesized by the modified RWP model to evaluate our protocol under different scenarios.

We evaluated the performance of PCS in connected MONs in comparison with several MANET replication algorithms: SAF [45], DCG [45], PDRS [33] and CACHE [91]. The details of these protocols can be found in Section 2.1.1.1. To better validate our analysis, we also compared PCS with Random, which places replicas on nodes randomly, and OPTM, which is a centralized protocol that calculates the ideal replica distribution based on our derived optimal replication rule. OPTM represents the best performance that can be obtained by the OFRR.

Table 3.2 shows the parameters used in experiments, unless otherwise specified. The parameters are determined by referring to the settings in [74, 91] and the real traces. According to the

works in [44,91], we determined the file size and storage space on each node. As the work in [58], the probability of originating requests for different files in each node follows a Zipf distribution and the Zipf parameter was set to 0.7. Initially, files were evenly distributed to each node and no replicas exist in the system. In the synthesized mobility, the speed of a node is randomly chosen from the range of $[s/2, 3s/2]$, in which s is the configured average node movement speed. Since the real trace does not indicate the communication range of each node, we set the communication range to 60m in the GENI experiment and 100m in the tests with the NS-2 to show the influence of different transmission ranges on the performance. We evaluated the performance of PCS with $K = 3$.

We used the following metrics in the experiments:

- *Hit Rate.* It is the percent of requests successfully resolved by either original files or replicas.
- *Average delay.* This is the average delay of all requests. To make the comparison fair, we included all requests in the calculation. For unresolved requests, we set their delays as the TTL.
- *Replication cost.* This is the total number of messages generated in creating replicates.
- *Cumulative Distribution Function (CDF) of the proportion of replicas.* This is the CDF of the proportion of replicas of each file. This metric reflects the amount of resources allocated to each file for replication.

Table 3.2: Simulation parameters.

	Real trace	**Synthesized mobility**
Environment Parameters	GENI / NS-2	NS-2
Simulation area	$600m \times 300m$	$1000m \times 1000m$
Node Parameters		
Number of nodes	35	60
Communication range	$60m$ / $100m$	$250m$
Average movement speed	-	$6m/s$
The size of a file (kb)	$1 - 10$	$1 - 10$
Number of files in each node	10	10
Storage space for replicas (kb)	50	50
Request Parameters		
Initialization period	$500s$ / $800s$	$200s$
Querying period	$1500s$ / $1200s$	$600s$
TTL of each request	$1000s$ / $200s$	$200s$
Total time for each test	$3000s$ / $3000s$	$1000s$

3.3.1 Performance in the GENI experiments with the Real Trace

3.3.1.1 Hit Rate and Average Delay

Table 3.3 shows the results of each protocol in the trace-driven experiments on GENI. We see that the hit rates in different replication protocols follow Random<CACHE <SAF<PDRS<DCG< PCS<OPTM and the average delays follow a reverse order: Random>CACHE>SAF> PDRS>DCG >PCS>OPTM. We see that OPTM and PCS lead to higher hit rate and lower average delay than others. This is attributed to the guidance of OFRR, which aims to minimize the average querying delay by considering both storage and meeting ability as resource to enhance overall file availability. PCS generates slightly lower hit rate and around 20% higher average delay than OPTM. This is because OPTM has the knowledge of all information needed in OFRR beforehand, while PCS has to create replicas in a fully distributed manner.

On the contrary, other protocols only replicate files locally, thereby creating redundant replicas and failing to achieve high file availability under node mobility. Random has the worst performance on hit rate and average delay. This is because Random only randomly creates replicas for files and fails to assign more resources to popular files, which are queried more frequently by nodes. CACHE only utilizes the storage on intersection nodes, which indicates that it fails to fully utilize storage space on all nodes. Therefore, it cannot create as many replicas as other protocols and exhibits a low hit rate and a high delay. In SAF, each node replicates its frequently queried files until its memory is filled up. Then, almost all resources are allocated to popular files. Therefore, SAF cannot optimize access delay globally. In PDRS, a node replicates files interested by its neighbors that have less storage resource than itself. However, as the replicas are not shared in the whole group, PDRS only renders a slightly performance improvement over SAF. DCG further improves SAF and PDRS by conducting the file replication on a group level. It eliminates duplicate replicas among group members and uses released memory for other replicas, thereby generating higher hit rate and smaller average delay than SAF and PDRS.

We find that the 1st percentiles of the delays of all protocols are 0.01. This is because some requests are immediately satisfied by direct neighbors. The 99th percentiles of the delays of the protocols approximately follow the relationship on average delay. Above results justify that PCS enhances the file searching efficiency by its global optimization of file availability. The fact that Random leads to worse performance than all methods that give priority to popular files when creating

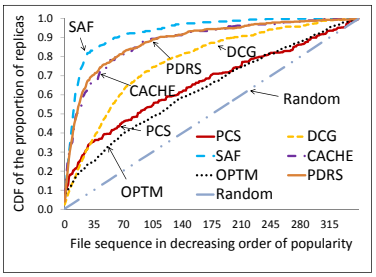


Figure 3.3: CDF of the resource allocated to replicas in trace-driven GENI experiment.

replicas also justify that a resource allocation strategy is necessary for file availability optimization.

Table 3.3: Experimental results of the trace-driven GENI experiment.

Protocol	Hit rate	Average / 1% / 99% delay (s)	Replication cost
Random	0.840139	263.176 / 0.01 / 991.9843	13387
CACHE	0.842454	260.469 / 0.01 / 994.2487	0
SAF	0.857341	259.1768 / 0.01 / 997.1095	0
PDRS	0.863074	256.1983 / 0.01 / 991.2384	175140
DCG	0.878559	251.3287 / 0.01 / 993.3947	67549
PCS	0.898823	240.7031 / 0.01 / 990.4522	28983
OPTM	0.910370	195.1776 / 0.01 / 990.1296	14542

3.3.1.2 Replication Cost

From the Table 3.3, we find that the replication costs of different protocols follow PDRS>DCG >PCS>OPTM≈ Random>SAF=CACHE=0. PDRS shows the highest replication cost because it needs to broadcast each new file to all nodes in the system. DCG incurs moderate replication cost because group members need to exchange information to reduce duplicate replicas. PCS has a low replication cost because each node only tries at most K times to create a new replica for each file it holds. OPTM and Random have a very low cost since nodes only need to communicate with the central server for replica list. SAF and CACHE have no replication cost since they do not need to exchange information among nodes for file replication. However, SAF generates a lot of redundant replicas, and Random and CACHE have a low performance, as shown in previous subsection.

3.3.1.3 Replica Distribution

Figure 3.3 shows the CDF of the proportion of resources allocated to each file for replica creation in different protocols. From the figure, we find that PCS exhibits the closest similarity to OPTM while other protocols follow: DCG≈Random≻CACHE≈PDRS≻SAF, where ≻ means closer similarity to OPTM. Combining the results on average delay, we find an interesting phenomenon:

except CACHE and Random, the protocol with closer similarity to OPTM has lower average delay. This proves the correctness of our theoretical analysis and the resultant OFRR rule expressed in Formula (3.22). CACHE has a low performance because it does not utilize all storage space, though it exhibits similarity with PDRS. Random creates replicas for each file randomly without considering their popularity, leading to a low performance since popular files are not replicated with priority. We also observe that the CDFs of the proportion of resource allocated to replicas of DCG, CACHE, PDRS and SAF increases to 0.9 quickly. This is because they allocate most resources to popular files, resulting in a lot of replicas for these files. Though these protocols can reduce the delay of requests for popular files, they cannot reduce the delay for unpopular files. PCS is superior over these protocols because it can globally reduce the average access delay of all files.

Table 3.4: Experimental results of the trace-driven NS-2 experiment.

Protocol	Hit rate	Average / 1% / 99% delay (s)	Replication cost
Random	0.828652	67.9564 / 0.00175637 / 193.259	4695
CACHE	0.830038	64.6417 / 0.00172859 / 191.703	0
SAF	0.837664	62.1525 / 0.00172887 / 190.896	0
PDRS	0.842982	61.0969 / 0.00172652 / 191.279	246454
DCG	0.848559	59.0611 / 0.00172883 / 189.270	14510
PCS	0.868749	50.2859 / 0.00172885 / 188.550	9846
OPTM	0.878677	41.2282 / 0.00172874 / 188.428	4721

3.3.2 Performance in the NS-2 Experiment with the Real Trace

3.3.2.1 Hit Rate and Average Delay

Table 3.4 shows the results of each protocol in the trace-driven experiments on NS-2. We see the hit rates and average delays of the seven protocols follow the same relationship as in Table 3.3 due to the same reasons. We find that the average delays of the seven protocols are much less than those in the GENI experiment. This is caused by two reasons. First, the trace-driven simulation adopts the PROPHET for file searching, which can locate files more quickly than the StaticWait searching protocol used in the GENI experiment. Second, the communication range of two nodes (100m) in the simulation is larger than that in the GENI experiment (60m), leading to shorter searching delay since a node can reach more neighbors. The hit rates of the seven protocols are lower than those in the GENI experiment. This is because the trace-driven simulation used a much smaller TTL. The relative performance between different protocols in the simulation matches with that in the GENI experiment, which further proves the effectiveness of PCS.

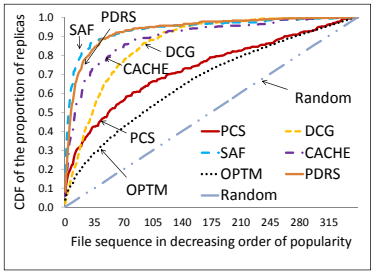


Figure 3.4: CDF of the resource allocated to replicas in trace-driven GENI experiment.

3.3.2.2 Replication Cost

From Table 3.4, we find that the replication costs of different protocols follow PDRS>DCG> PCS>OPTM ≈Random>SAF=CACHE=0. This matches with the results in Table 3.3 and the reasons are the same.

3.3.2.3 Replica Distribution

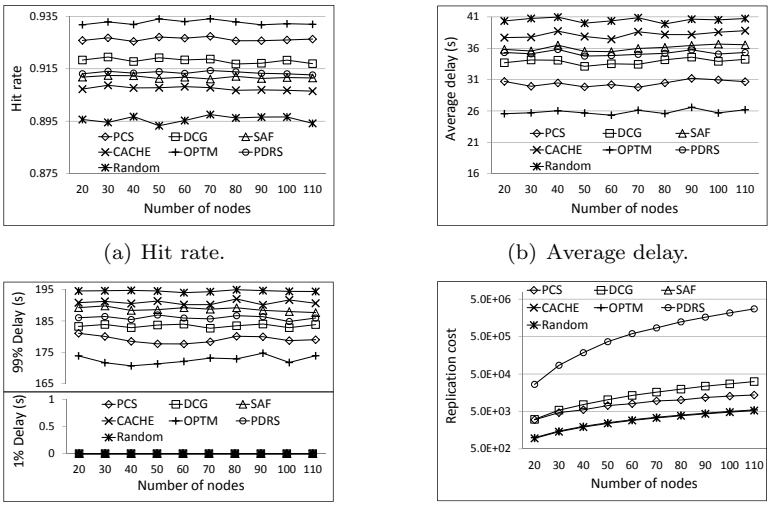
Figure 3.4 shows the CDF of the proportion of resource allocated to replicas of each file in the seven protocols. From the figure, we find similar trend as that in Figure 3.3. That is, except CACHE and Random, the protocol with closer similarity to OPTM has smaller average delay. This further proves the correctness of our analysis through the trace-driven simulation.

3.3.3 Performance in the NS-2 Experiment with Different Network Sizes

We examined the performance of PCS when the total number of nodes varied from 20 to 110 in the NS-2 simulator. In this test, node mobility traces were synthesized following the RWP mobility mode with different number of nodes, and the average node speed was set to 8 m/s. Other settings are the same as introduced in the beginning of Section 4.5.

3.3.3.1 Hit Rate

Figure 3.5(a) plots the hit rates of the seven replication protocols. We see the same relationship between different protocols as found in Table 3.3 and Table 3.4 with the same reasons. The reasons are also supported by the fact that the average delays of the seven protocols present reverse order of the hit rate, as shown in Figure 3.5(b).



(a) Hit rate.

(b) Average delay.

(c) The 1% & 99% delays.

(d) Replication cost.

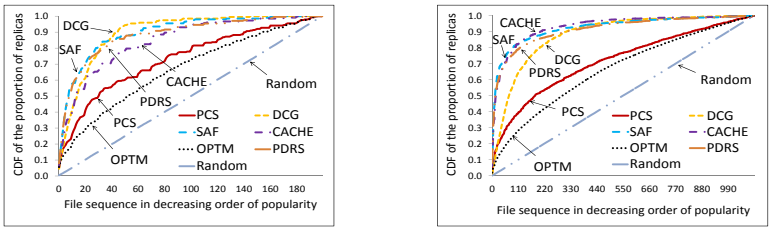
Figure 3.5: Performance of the file replication protocols with different network sizes.

3.3.3.2 Average Delay

Figure 3.5(b) shows the average request delays of the seven protocols. We observe the same results as that found in Table 3.3 and Table 3.4. Specifically, with different network sizes, PCS has 10%-15% less average delay than DCG, PDRS, SAF, CACHE, and Random and around 15% - 20% higher average delay than OPTM. Such results are consistent with aforementioned conclusions for the same reasons. The results in Figure 3.5(a) and Figure 3.5(b) confirm the validity of our analysis and the effectiveness of PCS in different network sizes.

More nodes in the network enables a node to have more neighbors and hence more options to forward requests to the file holders. It is interesting to see that in the figures, the hit rate and average delay of each protocol generally remain stable as the number of nodes increases. This is because the number of files is proportional to the number of nodes. More nodes does not increase the average storage space for replicas of each file. Therefore, the probability that a request forwarder meets a file holder remains approximately the same.

Figure 3.5(c) plots the 1st and 99th percentiles of the delays of the seven protocols. The relationship between the 99th percentiles delays of the seven protocols is in line with that in the average delays in Figure 3.5(b), Table 3.3, and Table 3.4 for the same reasons. The result confirms that PCS is effective in reducing the average querying delay in networks with different sizes.



(a) Network size = 20 nodes.

(b) Network size = 110 nodes.

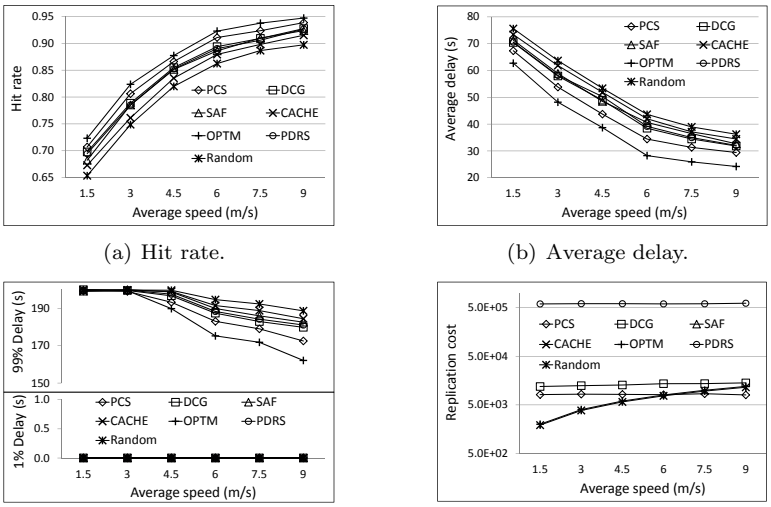
Figure 3.6: CDF of the resource allocated to replicas with different network sizes.

3.3.3.3 Replication Cost

Figure 3.5(d) illustrates the replication cost of each protocol. The replication costs of SAF and CACHE are not shown since they equal 0. We find that PDRS generates high replication cost, DCG shows moderate replication cost, and PCS, OPTM, and Random produce low replication cost. The result is consistent with those in Table 3.3 and Table 3.4 because of the same reasons. We also observe that the replication costs of PDRS, DCG, PCS, OPTM, and Random grow as the number of nodes in the system increases. This is because as the number of nodes increases, PDRS generates more messages during the broadcasting process for newly generated files, DCG produces more exchange messages between group members, PCS has more neighbors and more replication trials, and the central server in OPTM and Random needs to communicate with more nodes.

3.3.3.4 Replica Distribution

Figures 3.6(a) and 3.6(b) show the CDF of the proportion of resource allocated to replicas in each protocol when the number of nodes is 20 and 110, respectively. From the figures, we find that all protocols exhibit similar relationship as Figures 3.3 and 3.4. That is, except CACHE and Random, PCS shows the closest similarity to OPTM and others follow: DCG$\succ$PDRS$\succ$SAF. The results again confirm the correctness of our theoretical analysis with results from different network sizes. We find that Figures 3.6(a) and 3.6(b) show similar results, which demonstrates the correctness of OFRR in different network sizes. Combining above results, we conclude that OFRR can help shorten the average querying delay and PCS can realize it effectively in networks with different sizes.



(a) Hit rate.

(b) Average delay.

(c) The 1% & 99% delays.

(d) Replication cost.

Figure 3.7: Performance of the file replication protocols with different node mobility.

3.3.4 Performance with Different Node Mobilities

We examined the performance of PCS when the average speed varied from 1.5 m/s to 9 m/s in the NS-2 simulator. In this test, node mobility traces were synthesized following the RWP mobility mode with different average speed, and the number of nodes was set to a medium value of 60. Other settings are the same as introduced in the beginning of Section 4.5.

3.3.4.1 Hit Rate

Figure 3.7(a) shows the hit rates of the seven protocols. We find the same relationship on hit rates as those found in Table 3.3, Table 3.4, and Figure 3.5(a) with the same reasons. We also find that, for all protocols, the hit rate is low when nodes move slowly and is satisfactory (i.e., >90%) when the average speed is larger than 7.5 m/s. This is because nodes need longer time to encounter requested files when they move slowly, leading to more dropped requests due to TTL expiration.

3.3.4.2 Average Delay

Figure 3.7(b) shows the average querying delays of the seven protocols. We observe that PCS shows the closest result to OPTM, and it reduces the average delay of SAF, DCG, PDRS, CACHE, and Random by about 10%-15%. Again, the result is consistent with those in Table 3.3, Table 3.4 and Figure 3.5(b) due to the same reasons. The result also shows that the change of

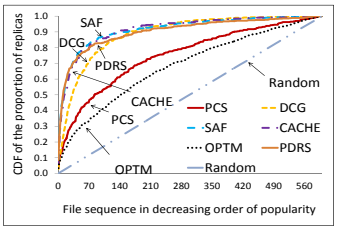
node movement speed does not affect the relative performance among different protocols. This is because, as shown in Equation (3.31), the effectiveness of replication protocol with the same network size and node mobility distribution is only determined by the resource allocation for file replicas. These results confirm the effectiveness of the PCS with different node mobility.

We also observe that the average delays of all protocols decrease as node movement speed increases. When nodes move faster, the average time needed for two nodes to meet is shortened, leading to shorter average delay. The result implies that the movement speed of a node affects the number of nodes it can encounter in a unit period and hence the availability of its files, which justifies the necessity of considering node meeting ability as a kind of resource in file replication.

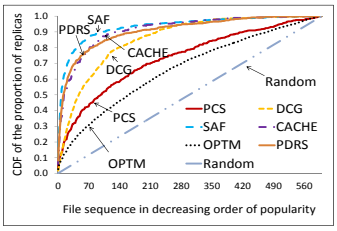
Figure 3.7(c) depicts the 1st and 99th percentiles of the delays of the seven replication protocols. Similar to the results in previous experiments, the 1st percentiles of delays of all protocols are nearly 0 and the 99th percentiles of the delays of these protocols present the same relationship as in Table 3.3 and Table 3.4 for the same reasons. When the average speed is slow (i.e., 1.5 m/s and 3 m/s), the 99th percentiles of the delays of all protocols equal the TTL (200s) since we use the TTL as the delay of dropped requests. When nodes move slowly, they need longer time to encounter requested files.

3.3.4.3 Replication Cost

From Figure 3.7(d), we find that PRDS presents the highest replication cost, DCG has moderate replication cost, and PCS, OPTM, and Random generate low replication cost. We did not plot the protocols with 0 cost. The relationship of the seven protocols on replication cost remains the same as in Table 3.3, Table 3.4, and Figure 3.5(d) because of the same reasons. We also find that the replication costs of PRDS, PCS and DCG remain stable when the node movement speed increases, while that of OPTM and Random increase. Since the DCG exchanges information between group members for replication and PRDS broadcasts messages through the network for newly created files, their replication costs are only decided by the number of nodes in the system. For PCS, the number of replica distribution attempts is not related to node mobility. Therefore, their replication costs remain stable when the node movement speed increases. In OPTM and Random, high mobility means more opportunities to meet the central server, leading to most communications and hence higher communication cost.

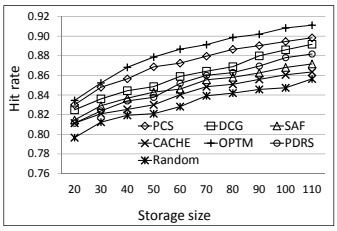


(a) Average speed = 1.5 m/s.

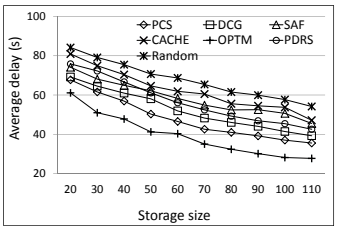


(b) Average speed = 9 m/s.

Figure 3.8: CDF of the resource allocated to replicas with different node mobility.



(a) Hit rate.



(b) Average delay.

Figure 3.9: Performance with different storage sizes.

3.3.4.4 Replica Distribution

Figures 3.8(a) and 3.8(b) show the CDF of the proportion of resource allocated to replicas of each file in each protocol when the average movement speed of nodes is 1.5 m/s and 9 m/s, respectively. We find that all protocols generate similar results as those in Figures 3.6(a) and 3.6(b) for the same reasons. The results again verify the correctness of our analysis and the effectiveness of PCS on following OFRR with various node mobility.

3.3.5 Performance with Different Storage Sizes

We also tested the performance of different replication protocols when the storage for replicas in each node increases from 20 to 110 on the NS-2 simulator. Figure 3.9 shows the hit rates and average delays of the seven protocols in the tests with the Dartmouth trace. We see that as the storage size increases, the hit rates of all protocols increase and their average delays decrease. This is because the number of replicas of each file increases when there is more storage space on each node, leading to higher hit rate and lower average delay. We also find that the performance relationship between the seven protocols in the test matches with those in Table 3.3 and Table 3.4 due to the same reasons. Such results further confirm the effectiveness of OFRR and PCS with different storage

resource limits.

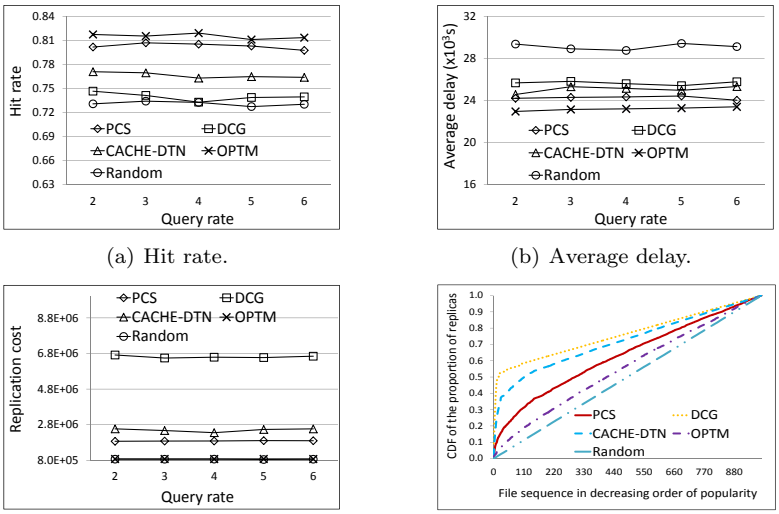
3.4 Performance Evaluation in Disconnected MONs based on the Community-based Mobility Model

In order to evaluate the performance of PCS in disconnected MONs, we conducted event-driven experiments with the MIT Reality project trace [34] and the Haggle project trace [18]. The MIT Reality trace lasts about 2.56 million seconds (Ms), while the Haggle project trace lasts about 0.34 Ms. Both traces represent typical MONs scenarios. We used the StaticWaiting routing protocol [87] in this test.

We evaluated the performance of PCS in comparison with DCG [45], CACHE-DTN [39], OPTM, and Random. CACHE-DTN caches each file on the central node of each network center location (NCL). If a central node is full, its replicas are stored in its neighbor nodes according to their popularities. A more popular replica is stored closer to the central node. The experiment settings and metrics are the same as in Section 3.3 unless otherwise specified in below. The total number of requests was set to $6000 * R_p$, and R_p is the request rate and was varied in the range of [2, 6]. In the experiment with the Haggle trace and the MIT Reality trace, requests were generated evenly in the time period of [0.3Ms, 2.3Ms] and [0.05Ms, 0.25Ms], and the TTL of each request was set to 0.3Ms and 0.04Ms, respectively.

3.4.1 Hit Rate

Figure 3.10(a) and Figure 3.11(a) plot the hit rates of the five methods with the Haggle trace and the MIT Reality trace, respectively. We see that in both scenarios, the hit rates follow OPTM>PCS>CACHE-DTN>DCG>Random. OPTM and PCS achieve higher hit rate than other methods because they follow the deduced OFRR. However, since PCS realizes OFRR in a distributed way, it presents slightly inferior performance compared to OPTM. CACHE-DTN considers the intermittent connection properties of MONs and replicates each file to every NCL, leading to high date accessibility, though not optimal. DCG only considers temporary connected group for data replication, which is not stable in MONs. Therefore, it has a low hit rate. Random assigns resources to files randomly, which means it cannot create more replicas for popular files, leading to the lowest



(a) Hit rate.

(b) Average delay.

(c) Replication cost.

(d) CDF of allocated resources.

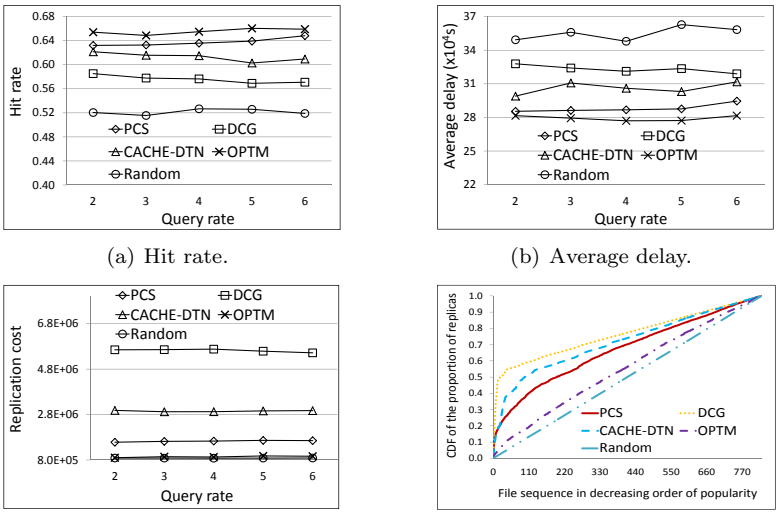
Figure 3.10: Performance of the file replication protocols with the Haggle trace.

hit rate. Such a result proves the effectiveness of the proposed PCS on improving the overall file availability and the correctness of our derived OFRR for MONs.

We also see that the hit rates of different methods fluctuate slightly when the request rate increases. This is because the hit rate is not affected by the request rate. Even when the number of request increases, the file availability remains on the same level and leads to similar hit rates, as shown in the two figures.

3.4.2 Average Delay

Figure 3.10(b) and Figure 3.11(b) demonstrate the average delays of the five methods with the Haggle trace and the MIT Reality trace, respectively. We find that in tests with both traces, the average delays follow OPTM<PCS<CACHE-DTN<DCG<Random, which is in reverse order of the relationship between the five methods on hit rate as shown in Figure 3.10(a) and Figure 3.11(a). This is because the average delay is reversely related to the overall data availability. As explained in above section, OPTM and PCS have high data availability since they follow OFRR; CACHE-DTN presents higher data availability than DCG because CACHE-DTN distributes every file to different NCLs while DCG only shares data among frequently encountered neighbor nodes; and all files in Random receive equal amount of resources for replicas. Such results further validate the proposed OFRR and PCS in disconnected MONs.



(a) Hit rate.

(b) Average delay.

(c) Replication cost.

(d) CDF of allocated resources.

Figure 3.11: Performance of the file replication protocols with the MIT Reality trace.

3.4.3 Replication Cost

Figure 3.10(c) and Figure 3.11(c) show the replication costs of the five methods with the Haggle trace and the MIT Reality trace, respectively. OPTM and Random have the lowest replication cost while the costs of other three methods follow PCS<CACHE-DTN<DCG. In OPTM and Random, nodes only need to contact the central server for replica list, leading to the lowest cost. DCG generates the highest replication cost since group members need to exchange a large amount of information to remove duplicate replicas. CACHE-DTN forwards each file to every NCL, leading to moderate replication cost. In PCS, a node tries at most K times to create a replica for each of its files, producing much lower replication cost than CACHE-DTN and DCG. Such a result demonstrates the high energy-efficiency of PCS. Combining all above results, we conclude that PCS has the highest overall file availability and efficiency compared to existing methods, and OFRR is effective in guiding optimal file replication in disconnected MONs.

3.4.4 Replica Distribution

Figures 3.10(d) and 3.11(d) show the CDF of the proportion of resources allocated to replicas in each protocol in the tests with the Haggle trace and the MIT Reality trace, respectively. We see in both figures, PCS present very close similarity to OPTM and the other two follow CACHE-DTN≻PCS. Random also present close similarity to OPTM on the CDF curve. However, the

difference between PCS and Random is that PCS assigns priority for popular files while Random gives even priority to all files. Since popular files are queried more frequently, Random still leads to a low performance. For other three methods that favor popular files, we find that the closer similarity with OPTM a protocol has, the better overall performance it has. Such a result also matches with what we observed in the tests for connected MONs. This proves the correctness of our theoretical analysis and the resultant OFRR rule for disconnected MONs.

3.5 Summary

In this chapter, we investigate the problem of how to allocate limited resources for file replication for the purpose of globally optimal file access efficiency in MONs. Unlike previous protocols that only consider storage as resources, we also consider file holder's ability to meet nodes as available resources since it also affects the availability of files on the node. We first theoretically analyze the influence of replica distribution on the average access delay under the constrained available resources with two representing mobility models, and then derive an optimal file replication rule that can allocate resources to file replicas for minimal average access delay. Finally, we design the Priority Competition and Split replication protocol (PCS) that can realize the optimal file replication rule in a fully distributed manner. Extensive experiments on both GENI testbed, NS-2, and event-driven simulator with real traces and synthesized mobility confirm both the correctness of our theoretical analysis and the effectiveness of PCS in MONs.

Chapter 4

Leveraging Social Networks for Efficient File Sharing in MONs

In this chapter, we expand the investigation to the scenario in which file requests can be actively forwarded among nodes to search for interested files. In this work, considering people in the same interest based social group tend to gather, we exploit social network properties to actively forward file requests to reach the file holders and forward the located file to its requester quickly, thereby realizing effective file sharing in MONs.

In the following, we first present the overview of the proposed SPOON system in Section 4.1. We then conduct trace analysis to verify social network properties in a real MON in Section 4.2. After this, we introduce the major components of SPOON in Section 4.3. We further present some discussions for more efficient file sharing in Section 4.4. In Section 4.5, the performance of SPOON is evaluated in comparison with other systems. Finally, Section 4.6 summarizes this chapter.

4.1 Overview

Recently, social networks are exploited to facilitate content dissemination/publishing in MONs [11,29,68,92]. These methods exploit the below property to improve the efficiency of message forwarding and content distribution.

- (P1) nodes (i.e., people) usually exhibit certain movement patterns (e.g., local gathering,

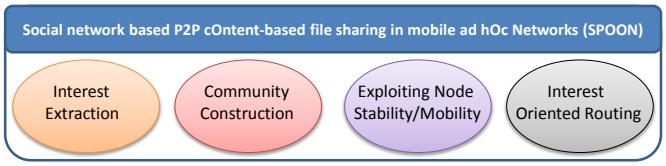


Figure 4.1: Components of SPOON.

diverse centralities, and skewed visiting preferences).

However, these methods are mainly designed for the dissemination of contents to subscribers and are not specifically designed for file sharing. They also fail to consider other properties of social networks revealed by recent studies to facilitate file sharing:

- (P2) Users usually have a few file interests that they visit frequently [37] and a user's file visit pattern follows a power-law distribution [55].
- (P3) Users with common interests tend to meet with each other more often than with others [75].

By leveraging these properties, we propose a Social network based P2P cOntent-based file sharing algorithm in mobile Opportunistic Networks (SPOON). SPOON has four main components, as shown in Figure 4.1.

(1) Based on P2, we propose an *interest extraction algorithm* to derive a node's interests from its files. The interest facilitates file requesting in content-based file sharing and other components of SPOON.

(2) We refer to a collective of nodes that share common interests and meet frequently as a *community*. According to P3, a node has high probability to find interested file in its community. If this fails, based on P1, the node can rely on nodes that frequently travel to other communities for file sharing. Thus, we propose the *community construction algorithm* to build communities to enable efficient file sharing.

(3) According to P1, we propose a *node role assignment algorithm* that takes advantage of node mobility for efficient file sharing. The algorithm designates a stable node that has the tightest connections with others in its community as the community coordinator to guide intra-community file sharing. For each known foreign community, nodes that frequently travel to it are designated as the community ambassadors for inter-community file sharing.

(4) We propose an *interest-oriented file searching and retrieval scheme* that utilizes an *interest-oriented routing algorithm (IRA)* and above three components. Based on P3, IRA selects request carrier by considering the probability of meeting keywords of interests rather than the probability of meeting nodes. The file searching scheme has two phases: intra- and inter-community searching. In the former, a node first requests nearby nodes, then relies on coordinator to search the entire home community. If it fails, the inter-community searching uses an ambassador to send the file request to a matched foreign community. A discovered file is sent back through the previous searching path or IRA if the path breaks.

SPOON is novel in that it leverages social network properties of both node interest and movement pattern. First, it classifies common-interest and frequently-encountered nodes into social communities. Second, it considers the frequency at which a node meets different interests rather than different nodes in file searching. Third, it chooses stable nodes in a community as coordinators and highly mobile nodes that travel frequently to foreign communities as ambassadors. Such a structure ensures that a request can be forwarded to the community of the requested file quickly.

4.2 Trace File Analysis

In order to validate the correlation between node interests and their contact frequencies, we analyzed the trace from the Haggle project [18], which contains the encountering records among 98 mobile devices carried by scholars attending the Infocom'06 conference. Some participants completed questionnaires to indicate the conference tracks that they are interested in.

We use T_t to denote the time length of the trace, and define the total meeting time of two nodes as the sum of the time length of each encountering. By regarding a community as a group of nodes in which each node has total meeting time larger than $T_t/4$ with at least half of all nodes in the community, we detected 8 communities from the trace. We then calculated each node's average number of shared interested tracks with other members in its own community C_i $(0 \leq i < 8)$, and with nodes in all other communities, respectively. Finally, the average values of all nodes in each community are calculated and shown in Table 4.1.

From the table, we see that for each community, nodes have higher average number of shared interested tracks with same community nodes than with nodes from other communities. Note that we used a relatively loose community creation requirement that each node only needs to have a

high contact frequency with half of nodes in a community. With a stricter requirement and a more sophisticated clustering method, nodes in the same community would share more interested tracks. Above results justify the previously mentioned social network properties and support the basis for SPOON that nodes with common interests tend to meet frequently.

Table 4.1: Average number of shared interested tracks.

Community C_i	Ave. # of shared interests with nodes in C_i	Ave. # of shared interests with nodes not in C_i
1	1.50	0.99
2	0.83	0.69
3	1.17	0.79
4	1	0.39
5	1.93	0.94
6	0.33	0.21
7	1.1	0.71
8	1	0.33

4.3 Main Components

A P2P file sharing system usually consists of 1) a method to represent contents, 2) a node management structure and 3) a file searching method based on 1) and 2). Accordingly, SPOON has four components to realize the three functions: 1) interest extraction, 2) community construction, 3) node role assignment, and 4) interest-oriented file searching and retrieval. We then present each component of SPOON.

4.3.1 Interest Extraction

Without loss of generality, we assume that node contents can be classified to different interest categories. It was found that users usually have a few file categories that they request for files frequently in a file sharing system. Specifically, for the majority of users, 80% of their shared files fall into only 20% of total file categories [37]. Like other file sharing systems [16,54], we consider that a node's stored files can reflect its file interests. Thus, SPOON derives the interests of a node from its files. Table 4.2 lists the notations used in this section.

To derive its interests, a node infers keywords from each of its files using the document clustering technique [84]. Specifically, a node derives a file vector for each of its files from its metadata. For file f_i, we denote its file vector by $v_i = (t_1, w_{it_1}; t_2, w_{it_2}; t_3, w_{it_3}; ...; t_m, w_{it_m})$, in which t_k and w_{ik} $(1 \leq k \leq m)$ denote a keyword and its weight that represents the importance

Table 4.2: Notations in interest extraction

Notation	Meaning
f_i and G_u	the i-th file and u-th interest group in a node
w_{it_k} and $\bar{w}_{ut_k}$	the weight of keyword t_k in f_i and in G_u
f_{ui}	the i-th file in G_u
v_i	the file vector of f_i
$\bar{v}_u$	the group vector of G_u
$\tilde{v}_N$	the node vector of node N

of the keyword in describing the file. We adopt the method in the text retrieval literature [12] to calculate the weight of a keyword, say t_k, in a file, say f_i, with below formula.

$$w_{it_k} = 1 + log(n_{t_k}), \tag{4.1}$$

where n_{t_k} refers to the number of occurrences of keyword t_k. Suppose there are m keywords in the file, we further normalize the weights by:

$$w_{it_k} = w_{it_k} / \sum_{q=1}^{m} w_{it_q}. \tag{4.2}$$

Then, in order to calculate the similarity of two file vectors, say v_1=$(t_1, w_{1t_1}; t_2, w_{1t_2}, t_4, w_{1t_4})$ and v_2=$(t_1, w_{2t_1}; t_3, w_{2t_3}; t_5, w_{2t_5})$, we first generate their *common vector*, which consists of their common keywords and corresponding weights in their own vectors. For example, the common vector of v_1 and v_2 is (t_1, w_{1t_1}) from v_1 and (t_1, w_{2t_1}) from v_2. We then use the following formula to calculate the similarity between v_1 and v_2:

$$sim(v_1, v_2) = \frac{\sum_{k=1}^{m'} w_{1_k} * w_{2_k}}{\sqrt{\sum_{k=1}^{m'} w_{1_k}^2} * \sqrt{\sum_{k=1}^{m'} w_{2_k}^2}} \tag{4.3}$$

where m' is the total number of common keywords and w_{1_k} and w_{2_k} represent the weights of the k^{th} common keyword of the two vectors, respectively.

After retrieving the file vector of each of its files, a node classifies its files to derive its interest groups. It creates a file similarity matrix $A = sim(v_r, v_s)$ ($1 \leq r$ & $s \leq \tilde{m}$), where $\tilde{m}$ is the number of files the node has. Since the similarities among files are known, we use the AGNES method [59] to cluster the files into interest groups in a hierarchical manner. Each file form an individual group

initially. Then, two most similar file groups are merged in each step. This process repeats until the similarity between any two groups is below a threshold. The similarity between two groups is calculated based on their interest vectors introduced below. Consequently, a file is classified to only one interest group and there is no overlap among groups.

Each group has a number of files. Suppose there are g files in interest group G_u, denoted by $(f_{u1}, f_{u2}, \ldots, f_{ug})$. The average weight of a keyword, say t_k, in the group is calculated by $\bar{w}_{ut_k} = \sum_{i=1}^{g} w_{t_k}^{f_{ui}}/g$, where $w_{t_k}^{f_{ui}}$ denotes the weight of t_k in f_{ui}. We also pre-define a threshold for the average weight, denoted by $T_{\bar{w}}$. We form an *interest vector* with keywords having weights larger than $T_{\bar{w}}$ and use it to represent interest group G_u :

$$\bar{v}_u = (t_1, \bar{w}_{ut_1}; t_2, \bar{w}_{ut_2}; t_3, \bar{w}_{ut_3}; ...; t_{n'}, \bar{w}_{t_{un'}}). \tag{4.4}$$

where n' is the total number of keywords in $\bar{v}_u$. Thus, each node has a number of interest vectors to represent its interests. The weight of G_u, denoted by $W(G_u)$, is the portion of the group's files in all files of the node. We then generate a *node vector* $(\tilde{v}_N)$ to describe a node's interests. The keywords of $\tilde{v}_N$ is the keyword union of all interest group vectors, and the weight of each keyword is the sum of its weights in different interest groups it belongs to normalized by the weights of these groups.

4.3.2 Community Construction

Social network theory reveals that people with the same interest tend to meet frequently [75]. By exploiting this property, SPOON classifies nodes with common interests and frequent contacts into a community to facilitates interest based file searching, as introduced latter in Section 4.3.4. Nodes with multiple interests belong to multiple communities. The community construction can easily be conducted in a centralized manner by letting a central node collect node interests and contact frequencies from all nodes. However, considering that the proposed system is for MONs, in which timely information collection and distribution is non-trivial, we further propose a decentralized method to ensure the adaptivity of SPOON in real environment.

When two nodes, say N_1 and N_2, meet, they consider two cases for community creation: (1) they do not belong to any communities, and (2) at least one of them is already a member of a community. In the first case, they calculate the similarity between each pair of their interest

vectors using Formula (4.3). A pair of interest groups, say G_i and G_j with interest vectors $\bar{v}_i$ and $\bar{v}_j$, is called *matched interest group* when $W(G_i)W(G_j)sim(\bar{v}_i, \bar{v}_j) \geq T_G$, where T_G is a predefined threshold. The purpose of taking into account the weight of each interest group is to eliminate the noise of interest groups with a small number of files and achieve better interest clustering. If N_1 and N_2 have at least one pair of matched interest group, and their contact frequency, $F(N_1, N_2)$, is higher than the top $h_1\%$ highest encountering frequencies in either node, the two nodes form a new community. The keywords in their matched interest groups and corresponding weights constitute the *community vector (v_C)* of the community.

In the second case, suppose N_2 is already a member of community C, N_1 then calculates $W(G_i)sim(\bar{v}_i, v_C)$ for each of its interest groups, say G_i, to decide whether it should join in community C. If the similarity value for one interest group is larger than T_G, and N_1's contact frequency with community C is higher than the top $h_2\%$ of N_1's contact frequencies with all nodes it has met, N_1 is granted the membership to community C. The contact frequency with community C refers to the accumulated contact frequency with nodes in C. It is updated upon each encountering with a node in C. This means that N_i contacts members in community C frequently enough to guarantee connections. N_1 then copies the community vector and other community information from N_2. Also, when a node meets the community coordinator, it reports its files to the coordinator to update its file index and community vector. The coordinator then forwards the updated community information to community members when meets them.

With above community construction method, nodes with common interests and frequent contacts gradually form a community. However, nodes that appear later have more stringent community acceptance requirement. Its contact frequency to the community needs to be higher than that of more nodes, and its interest vector is compared with a longer community vector. Also, nodes in a group admit new members distributively. As a result, nodes in a group may not have very similar interests or high contact frequencies. We propose two solutions to alleviate this problem. First, we set an initial period for newly joined nodes to enable them to accumulate contact frequencies with others. Then, when a node starts to join in communities, its meeting frequencies with others are relatively stable, which provides more accurate measurement for determining the communities to join in. Second, we use group member pruning. Existing community members can have a second round voting to confirm the eligibility of new community members. Specifically, if N_2 in community C finds a node, say N_1, satisfies the requirements of C, it awards N_1 a potential membership for C.

Then, other community members in C further checks N_1's eligibility to join in C. That is, every time when N_1 meets an existing member of C, say N_3, N_3 checks whether they have at least one pair of matched interest group and whether their contact frequency is higher than the top h_1% of N_1's highest contact frequencies. If yes, N_3 approves the membership of N_1. When an existing community member of C notices that N_1 receives the grants from half of the community members, it grants N_1 the group membership.

Another issue is that node contact frequencies and interests may change over time. Since the community construction algorithm is continuous running, a node can detect that it fails to satisfy the requirement of current community. It then withdraws from the current community, notifies connected nodes in it, and search for a new community to join in.

The values of the thresholds used in the community construction process (T_G, h_1, h_2) are determined by many factors such as number of nodes, number of interests, and types of applications. Generally, T_G decides the interest tightness among nodes in each community. A larger T_G leads to higher similarity between interests of nodes in one community, but also generates more communities. Therefore, T_G should be configured based on application scenario. If the application has clear file categories (i.e., course file sharing), we can set a large T_G to gather files in the same category. Otherwise, a medium T_G should be set to balance the interest closeness and the number of communities. h_1 and h_2 determines the tightness of a community. The smaller h_1 and h_2 are, the tighter the community is. Therefore, we set them to 30 by default in experiment to ensure frequent contact among community members.

4.3.3 Node Role Assignment

A previous study has shown that in a social network consisting of mobile users, only a part of nodes have high degrees [92]. We can often find an important or popular person who coordinates members in a community in our daily life. For example, the college dean coordinates different departments in the college, and the department head connects to faculty members in the department. Thus, we take advantage of different types of node mobility for file sharing.

We define community coordinator and ambassador nodes in the view of social network. A community coordinator is an important and popular node in the community. It keeps indexes of all files in its community. Each community has one ambassador for each known foreign community, which serves as the bridge to the community. The coordinator in a community maintains the v_C of

foreign communities and corresponding ambassadors in order to map requests to ambassadors for inter-community searching. The number of ambassadors and coordinators can be adjusted based on the network size and workload in order to avoid overloading these nodes. Since ambassadors and coordinators take more responsibility, we can also adopt role rotation and extra incentives for fairness consideration.

4.3.3.1 Community Coordinator Node Selection

We define a stable node that has tight contact frequency with other community members as the community coordinator. In network analysis, centrality is often used to determine the relative importance of a vertex within the network. We then adopt the improved degree centrality [30], which assigns weight to each link based on the contact frequency, for coordinator selection since it reflects the tightness of a node with other community members. In the initial phase of coordinator discovery, each node, say node N_i, in a community collects contact information from its neighbors in the same community and then calculates its degree centrality by

$$D(p_i) = \sum_{j=1, j\neq i}^{N} w_{ij}, \tag{4.5}$$

where w_{ij} is the link weight between N_i and N_j and N is the number of neighbors in the same community. In order to reflect the property that the coordinator has the most connections with all community members, w_{ij} equals 1 if the contact frequency between N_i and N_j is larger than a threshold and 0 otherwise. Though such a method cannot ensure its connection to every community member, it ensures that the coordinator has the tightest overall connection to all community members.

Each node periodically checks its degree centrality and broadcasts such information to all community members. If a node receives no larger centrality score than its own centrality for three consecutive periods, it claims itself as the potential coordinator. The potential coordinator would confirm its status as the coordinator when meets the previous one. If it is confirmed, it then requests the community information from the old coordinator. Also, when the new coordinator meets community members, they exchange information for group vector update and ambassador selection, as well as file request routing.

4.3.3.2 Community Ambassador Node Selection

An ambassador is used to bridge the coordinator in its home community and a foreign community. We use the product of a node's contact frequency with its coordinator and that with the foreign community for ambassador selection. Each node i calculates its utility value for foreign community k by

$$U_{ik} = F(N_i, C_k) * F(N_i, N_c) \tag{4.6}$$

where C_k represents foreign community k, N_c is the coordinator in its home community, and $F(\cdot)$ denotes the meeting frequency. Each node reports its utility values for foreign communities it has met to the coordinator in its home community. Then, the community coordinator chooses one ambassador for each known foreign community. Also, the node that has the highest overall contact frequency with all foreign communities is selected as the default ambassador. In case that a request fails to find a matched ambassador, the default ambassador can carry the request and seek for potential forwarders in foreign community. If an ambassador loses the connection with the coordinator for a certain period of time, a new ambassador that satisfies above requirements is selected. This arrangement facilitates interest-oriented file searching by enabling a coordinator to send file requests to matched foreign communities quickly.

In above design, ambassadors are the key to connect different communities efficiently. Coordinators achieves balance between the centralized and distributed searching by checking whether a community can satisfy a request quickly, which is important in MONs. Also, though broadcast is used in coordinator selection, the cost is limited because 1) it is only among community members, and 2) we can set a long inter-broadcast period since nodes usually have stable degree centrality. To select ambassadors, each node just reports its utility values to the coordinator, which can be piggybacked on the beacon messages. Therefore, this step does not incur significant extra costs.

4.3.4 Interest-oriented File Searching and Retrieval

In social networks, people usually have a few file interests [37] and their file visit patterns generally follow a certain distribution [55]. Also, people with the same interest tend to contact each other frequently [75]. Thus, interests can be a good guidance for file searching. Considering the relation among node movement pattern [50], individuals' common interests, and their contact frequencies, we can route file requests to file holders based on nodes' frequencies of meeting different

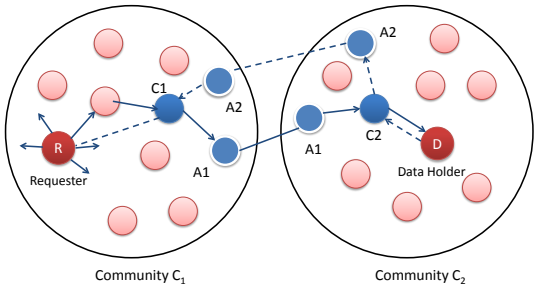


Figure 4.2: File searching in SPOON.

interests.

Then, the interest-oriented file searching scheme has two steps: intra-community and inter-community searching. A node first searches files in its home community. If the coordinator finds that the home community cannot satisfy a request, it launches the inter-community searching and forwards the request to an ambassador that will travel to the foreign community that matches the request's interest. A request is deleted when its TTL (Time To Live) expires. During the search, a node sends a message to another node using the *interest-oriented routing algorithm (IRA)*, in which a message is always forwarded to the node that is likely to hold or to meet the requested keywords. The retrieved file is routed along the search path or through IRA if the path expires.

4.3.4.1 Interest-oriented Routing Algorithm

In SPOON, every node maintains a history vector that records its frequency of encountering interest keywords. The history vector is in the form of $v_H = (t_0, w_{h0}; t_1, w_{h1}; t_2, w_{h2}; ...; t_n, w_{hn})$, where w_{hi} is the aggregated times of encountering keyword t_i. w_{hi} decays periodically as time passes by $w_{hi} = \gamma w_{hi} (\gamma < 1)$. When two nodes meet, they exchange their node vectors and update history vectors. The history vector is used to evaluate the probability of meeting the requested content.

The destination of a request is represented by a vector $v_{dest} = (t_0, w_0; t_1, w_1; t_2, w_2; ...; t_n, w_n)$. In IRA, a node uses the fitness score $\mathcal{F}$ to evaluate its neighbors' probabilities to be or to meet the file holder. The fitness $\mathcal{F}$ of neighbor i is measured by $\mathcal{F} = \alpha sim(v_{dest}, \tilde{v}_i) + (1 - \alpha) sim(v_{dest}, v_{Hi})$, where $\tilde{v}_i$ and v_{Hi} are the node vector and history vector of node i, respectively. The factor of $sim(v_{dest}, \tilde{v}_i)$ aims to find the node sharing the most similar interests with the destination, and the factor of $sim(v_{dest}, v_{Hi})$ aims to find a node that is very likely to meet the destination in its movement. α is used to control the weight of these two factors. In IRA, when a node receives a

message, if its neighbor with the highest $\mathcal{F}$ has higher $\mathcal{F}$ than itself, it forwards the message to the neighbor. This process repeats until the message arrives at the destination. Coordinators do not use IRA but send messages to its community members when meeting them because they usually have tight connections with all community members.

4.3.4.2 Intra-Community File Searching and Retrieval

The file request message is represented by a request vector represented as:

$$v_Q = (t_0, w_0; t_1, w_1; t_2, w_2; ...; t_n, w_n). \tag{4.7}$$

Each request is associated with a counter (*count*) indicating the number of hops it can travel. The *count* is decremented by 1 after each forwarding. Since the request is initiated by users, term weights in v_Q are constant values. In the intra-community searching, the destination that a request is sent to is represented by a combination of the v_Q and the node vector of the requester's community coordinator (v_{N_C}), represented by:

$$v_{dest} = \lambda v_Q + (1 - \lambda) v_{N_C}, \tag{4.8}$$

In the first step, the requester calculates the similarity between the *request vector* and the *community vector* of the community it belongs to. If $Sim(v_Q, v_C) < T_s$, the request is sent to the coordinator of the community directly (i.e., λ equals 0). Otherwise, λ equals 1 when the counter (*count*) is larger than 0 and 0 otherwise. This means that a requester first searches nearby nodes within *count* hops, and then resorts to its community coordinator. Specifically, the requester sends out a request to top π neighbors with the highest $\mathcal{F}$. Having $\pi > 1$ copies of a request can enhance the efficiency of file searching. We call this strategy *multi-copy forwarding*. In order to limit the number of copies for each request, we set $\pi = min\{k | \prod_{i=0}^{k} \mathcal{F}_i > \beta\}$, where $\mathcal{F}_i$ is the fitness of neighbor i and β is the minimum delivery guarantee factor. The hop counter of a request is decreased by one after each forwarding. If the file is not found when $count = 0$, it is forwarded to the community coordinator ($\lambda = 0$). When a node receives multiple copies of the same requests, it only processes the first one.

When node N_j receives a request, if $v_{dest} = v_Q$ and $sim(v_{dest}, v_{N_j})$ reaches the similarity threshold specified by the file requester, it first tries to send the satisfied files back to the requester

along the original path. If a forwarder on the path is not available due to node mobility, IRA is used to forward the file. Otherwise, N_j uses IRA to further forward the request. If $v_{dest} = v_{N_c}$ and N_j is not the coordinator N_c, N_j uses IRA to forward the request to N_c. After N_c receives the request, it checks its file indexes. If the indexes have files satisfying the request, the coordinator sends the request to the file holder when meeting it, which then sends the file back to the requester. Otherwise, N_c initiates the inter-community file searching. Algorithm 1 shows the pseudocode of the intra-community searching algorithm.

Algorithm 1:Pseudo-code of intra-community file searching for request Q conducted by node N_i.

```
Procedure intraSearchForQ ()
  if a neighbor nb of N_i matches request Q then
      N_i.sendQeuryTo(Q, nb)
  else if Q.src = N_i then
      if Sim(v_Q, v_C) < T_s then
          Q.v_dest ← v_N_C
          N_i.sendThroughIRATo(Q, N_C)
      else
          Q.v_dest ← v_Q
          N_i.rankNbByFitness()
          overallF ← 0
          for each neighbor nb of node N_i do
              overallF gets overallF + F(Q, nb)
              N_i.sendQeuryTo(Q, nb)
              if overallF > β then
                  break
  else
      if Q.hops < MaxHopthen
          Q.v_dest ← v_Q
          N_i.rankNbByFitNess()
          nb ← the neighbor with maximal fitness
          N_i.sendQeuryTo(Q, nb)
      else
          Q.v_dest ← v_N_C
          N_i.sendThroughIRATo(Q, N_C)
```

4.3.4.3 Inter-Community File Searching and Retrieval

In the inter-community searching algorithm, a coordinator maps a request to the foreign community that is most likely to contain the requested file. Similar to the intra-community search

step, the coordinator also uses the multi-copy forwarding strategy, i.e., it sends out a request to Ω ambassadors having the highest similarity with the request in order to enhance the efficiency of the forwarding. We limit the number of copies for each request by letting $\Omega = min\{k|\prod_{i=0}^{k} sim(v_Q, v_C) > \alpha\}$, where α is the minimum delivery guarantee factor. Ambassadors then forward the request to the corresponding foreign communities.

Upon receiving the request, the coordinator in the foreign community checks its file indexes to see if its community has the file. If not, the coordinator repeats the inter-community file searching by looking up its ambassadors to check for further forwarding opportunities. If the file exists, the coordinator asks for the file from the file holder when meeting it and sends the file back to the requester's community through the corresponding ambassador. The coordinator of the requester's community will further forward the file to the requester.

Figure 4.2 depicts the process of file searching, in which a requester (node R) in community C_1 generates a file request. Since its neighbors within *count* hops don't have the file, the request is then forwarded to the community coordinator N_{C1}. N_{C1} checks the community file indexes but still can't find the file. It then asks the community ambassador N_{A1} to forward the request to the foreign community matching the requested file. Using the same way as N_{C1}, the community coordinator N_{C2} finds the file and sends it back to the requester's community via ambassador N_{A2}. The file is first sent to N_{C1}, and then forwarded to the requester. Algorithm 2 shows the pseudocode of the inter-community searching algorithm.

4.3.5 Information Exchange among Nodes

We summarize the information exchanged among nodes in SPOON. In the community construction phase, two encountered nodes exchange their interest vectors and community vectors, if any, for community construction. In the role assignment phase, nodes broadcast their degree centrality within their communities for coordinator selection. When the coordinator is selected, the coordinator ID is also broadcasted to all nodes in the community. Then, each node reports its contact frequencies with foreign communities to the coordinator for ambassador selection. Besides, when a node meets a coordinator of its community, the node also sends its updated node vector to the coordinator to update the community vector and retrieves the updated community vector from the coordinator. When an ambassador meets the coordinator of its community, it reports the community vectors of foreign communities to the coordinator. After above information exchange,

Algorithm 2:Pseudocode of inter-community file searching for request Q conducted by node N_i.

```
Procedure interSearchForQ ()
  if N_i is a coordinator then
      bContain ← N_i.checkContainFile(Q)
      if bContain
          N_i.sendQeuryToDes(Q)
      else
          N_i.rankAmByMatch()
          overallS ← 0
          for each ambassador N_A of N_i's community do
              n.sendQeuryTo(q, N_A)
              overallS ← overallS + Sim(q.V_Q, N_A.V_C)
              if overallS > α
                  break
  if N_i is an ambassador then
      when N_i meets another node N_j
      if N_j.homeCommunity = N_i.foreighCommunity then
          N_i.sendQeuryTo(Q, N_j)
          N_j.sendThroughIRATo(Q, N_C)
```

two encountered nodes exchange their node vectors and history vectors for packet routing. Each node checks packets in it sequentially to decide which packets should be forwarded to the other node based on the file searching algorithm introduced in Section 4.3.4. Further, when network turns to be stable, the frequency of information exchange for community construction and node role assignment can be reduced to save costs.

4.4 Discussion on Advanced Methods

We further present some advanced schemes than can help improve the file sharing efficiency in this section.

4.4.1 Intelligent File Prefetching

Ambassadors in SPOON can meet nodes holding different files since they usually travel between different communities frequently. Taking advantage of this feature, an ambassador can intelligently prefetch popular files outside its home community. Recall that a request in a local

community for a file residing in a remote community are forwarded through the coordinator of the local community. Thus, each coordinator keeps a track of the frequency of local requests for remote files and provides the information of popular remote files to each ambassador in its community upon encountering it. When a community ambassador finds that its foreign community neighbors have popular remote files that are frequently requested by its home community members, it stores the files on its memory. The prefetched files can directly serve potential requests in the ambassador's home community, thus reducing the file searching delay.

4.4.2 Request-Completion and Loop-Prevention

Given a file request, there may exist a number of matching files in the system. A node can associate a parameter S_{max} with its request to specify the number of files that it wishes to find. A challenge we need to handle is to ensure that the file searching process stops when S_{max} matching files are discovered when multi-copy forwarding is used. To solve this problem, we let a request carry S_{max} when it is generated. When a request finds a file that matches the request and is not discovered before, it decreases its S by 1. Also, if this request is replicated to another node, S is evenly split to the two nodes. A request stops searching files when its S equals 0.

When a request needs to find more than one file, it is likely that IRA would forward a request to the same node repeatedly. To avoid this phenomenon, SPOON incorporates two strategies. First, the request holder inserts its ID to the request before forwarding the request to the next node. Second, a node records the requests it has received within a certain period of time. The former method avoids sending a request to nodes it has visited before while the latter method prevents sending different replicas of the same request to the same node. Specifically, when a node, say N_i, needs to forward a request to a newly met node N_j based on IRA, N_i checks whether the request's record of traversed nodes contains N_j. If yes, N_i does not forward the request to N_j. Also, when a node receives a request, if the request exists in its record of received requests, the node sends the request back to the sender. These two strategies effectively avoid searching loops by simply preventing a node from forwarding the same request to nodes that have received the request before.

4.4.3 Node Churn Consideration

In SPOON, when a node joins in the system, it first finds the communities it belongs to and learns the IDs of community coordinators, and then reports its files and utility values to the community coordinator when encountering it. This enables the coordinator to maintain updated information of the community members.

A node may leave the system voluntarily when users manually stop the SPOON application on their devices. In this case, a leaving node informs its community coordinator about its departure through IRA. If the leaving node is an ambassador, the coordinator then chooses a new ambassador. If the leaving node is a coordinator, it uses broadcast to notify other community members to select a new coordinator.

A node may also leave the system abruptly due to various reasons. Simply relying on the periodical beacon message, a node cannot tell whether a neighbor has left the system or just moves away, which is a usual case in MONs. To handle this problem, each node records the timestamps when it meets other nodes, and sends it to the coordinator through IRA. The coordinator receives this information and updates the most recent timestamp of each node seen by other nodes. If the coordinator finds that a node's timestamp is more than T_x seconds ago, it considers this node as a departed node. Similarly, normal nodes in a community also maintain and update the timestamp of the coordinator to determine whether it is still alive. A node piggybacks the coordinator departure information on the beacon messages. Then, its nearby nodes can know whether the coordinator has left. Note that a node can know the number of community members from the coordinator. When a node finds that more than half of community members have found that the coordinator has left, it broadcasts a coordinator re-election message to select a new coordinator using the same method explained in Section 4.3.3.1.

4.5 Performance Evaluation

We evaluated the performance of SPOON in comparison with MOPS [68], PDI+DIS [71,82], CacheDTN [39], PodNet [66], and Epidemic [89]. MOPS is a social network based content service system. It forms nodes with frequent contacts into a community and selects nodes with frequent contacts with other communities as brokers for inter-community communication. PDI+DIS is a combination of PDI [71] and an advertisement-based DISsemination method (DIS) [82]. PDI provides

distributed search service through local broadcasting (3 hops), and builds content tables in nodes along the response path, while DIS let each node disseminate its contents to its neighbors to create content tables. CacheDTN replicate files to network centers in decreasing order of their overall popularity. In PodNet, nodes cache files interested by them and nodes they have met. We adopted the "Most Solicited" file solicitation strategy in PodNet. We doubled the memory on each node in CacheDTN and PodNet for replicas. In Epidemic, when two nodes meet each other, they exchange the messages the other has not seen. We have conducted the following experiments.

(1) *Evaluation of Community Construction.* We first evaluated the proposed community construction algorithm introduced in Section 4.3.2.

(2) *GENI experiments.* We conducted experiments on the GENI ORBIT testbed [3, 4] using the MIT Reality trace. The GENI ORBIT testbed contains 400 nodes with 802.11 wireless cards. Nodes can communicate with each other through the wireless interface. We used the real trace to simulate node mobility in ORBIT: two nodes can communicate with each other only during the period of time when they meet in the real trace.

(3) *Event-driven experiments with real traces.* We also conducted event-driven experiments with two real traces.

(4) *Evaluation of enhancement strategies.* We evaluated the effect of the enhancement strategies introduced in Section 4.4.1, 4.4.2, and 4.4.3 through event-driven experiments.

We disabled the intelligent prefetching and the multi-copy forwarding (i.e., we set $\pi = 1$ and $\Omega = 1$) in SPOON to make the method comparable to other methods. Also, since the comparison methods can return only one file for a request, we set $S_{max} = 1$ in SPOON. In each community, we used the node having the most contacts with other communities as the ambassador in SPOON and as the broker in MOPS. We also set the node with the most contacts with its community members as the coordinator in SPOON.

Besides the Haggle trace, we further tested with the MIT Reality trace [34], in which 94 smart phones were deployed among students and staffs at MIT to record their encountering. The two traces last 0.34 million seconds (Ms) and 2.56 Ms, respectively. As in MOPS, we used 40% of the two traces to detect groups in which nodes share frequent contacts. Here, we use "group" to represent a group of nodes with frequent contacts, and use "community" to represent a group of

nodes with common-interests and frequent contacts. We got 7 and 8 groups for the MIT Reality trace and the Haggle trace, respectively. Then, since there is no real trace for P2P file sharing over MONs, we collected articles from different news categories (e.g., sports, entertainment and technology) from CNN.com and mapped them to the identified communities. Each node contains 50 articles from the news category for its community. Each node extracts its interests from its stored files. The similarity threshold was set to 70% in AGNES for file classification.

In experiments with the Haggle trace and the MIT Reality trace, we set the initialization period to 0.09Ms and 0.3Ms, the request generation period to 0.1Ms and 1Ms, and the TTL of a request to 0.15Ms and 1.2Ms, respectively. Considering that people usually generate requests according to their interests, we set 70% of total requests searching for files located in the local communities of their requesters. Each request is for an article randomly selected from the article pool. We measured following metrics:

(1) **Hit rate**: the percentage of requests that are successfully delivered to the file holders.

(2) **Average delay**: the average delay of the successfully delivered requests.

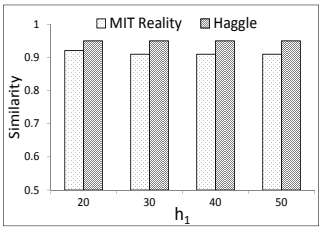
(3) **Maintenance cost**: the total number of all messages except the requests, which are for routing information establishment and update or replication creation.

(4) **Total cost**: the total number of messages, including maintenance messages and requests, generated in a test.

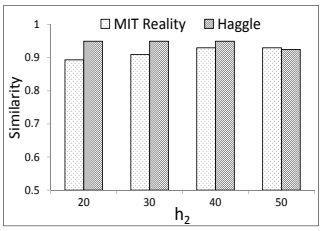
4.5.1 Evaluation of the Community Construction

We first tested the effectiveness of the community construction method in SPOON, denoted by SPOON-CC, in comparison with Active-CC and Centralized-CC. The Active-CC selects three most active nodes to collect node contacts and interests when they meet nodes. In Centralized-CC, we purposely let a super node collect all node contacts and interests timely. Both Active-CC and Centralized-CC use AGNES to build communities with the collected information.

Since there is no real trace about P2P file sharing in MONs, we tested in an indirect way. We first conducted the group construction and content distribution as previously described, and then removed the group identity of each node. Then, we run the three methods to create communities. After this, we matched each new community to the most similar old community and calculated the



(a) $h_2 = 30$ and $h_1 = 20 - 50$.



(b) $h_1 = 30$ and $h_2 = 20 - 50$.

Figure 4.3: Average similarity values with different h_1 and h_2.

similarity value by $N_{sm}^2/(N_p * N_n)$, where N_{sm} is the number of common nodes, N_p and N_n denote the size of the old community and the new community, respectively. In SPOON-CC, we set T_G to 1 to ensure interest closeness, and set h_1 and h_2 to 30. Active-CC and Centralized-CC used the same threshold for granting community membership as SPOON-CC. The average similarities in SPOON-CC, Active-CC, and Centralized-CC are 0.95, 0.87, and 1 with the Haggle trace and 0.91, 0.85, and 1 with the MIT trace, respectively. The Active-CC has inferior performance since active nodes can only collect information from nodes they have met, leading to less accurate community construction. Also, the performance of SPOON-CC is close to that of Centralized-CC, which has the best performance. Such a result shows the effectiveness of SPOON-CC in this test.

We further varied h_1 and h_2 from 20 to 50 to verify the effectiveness of SPOON-CC. Figure 4.3(a) and 4.3(b) show the average similarity values with the two traces. We see that the similarity values are above 90% with various h_1 and h_2. Such results further confirm the effectiveness of SPOON's community construction algorithm to cluster nodes with frequent contacts and similar interests in the experiments with the two real traces.

4.5.2 GENI Experiments

Table 4.3 shows the results of the GENI experiments of the six methods. From the table, we find that Epidemic generates the highest hit rate with the highest total cost and a low average delay. This is resulted from the dissemination nature of broadcasting. SPOON produces the second highest hit rate at the second lowest total cost and relatively high average delay. This is because SPOON utilizes both contact and content properties of social networks to guide file searching. Therefore, it can successfully locate requested files without the need of many information exchanges and request messages, though at a relatively slow speed. SPOON outperforms MOPS in terms of hit rate, delay,

and cost. This is because SPOON utilizes IRA for intra-communication and dedicated ambassadors for inter-communication while MOPS relies heavily on brokers. Also, MOPS only considers node contact in routing, while SPOON considers both content and contact. We will elaborate the reasons in describing the trace driven simulation results later on.

Table 4.3: Efficiency and cost in the experiments on GENI

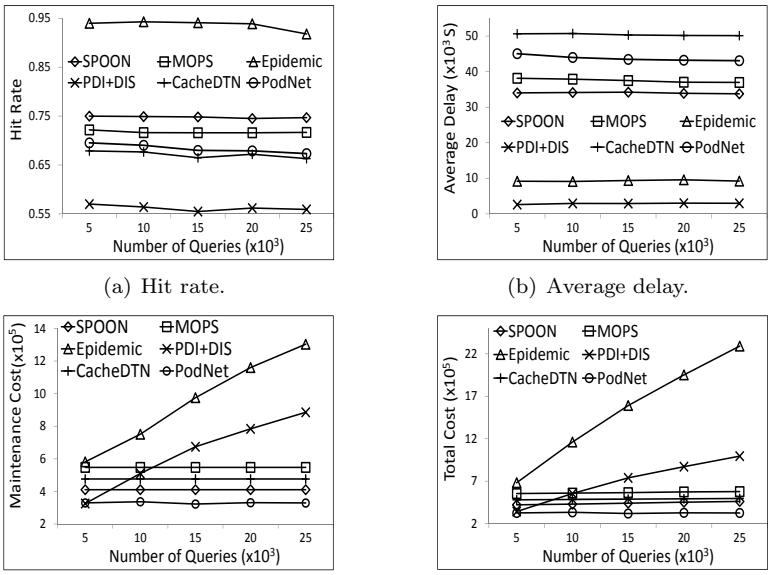
Method	Hit Rate	Ave. Delay (s)	Maintenance Cost	Total Cost
SPOON	0.671	152731.3	258764	275312
MOPS	0.629	163282.5	310131	320412
CacheDTN	0.5712	219021.4	283210	298123
PodNet	0.5932	183621	223218	240238
PDI+DIS	0.524	7418.9	298641	359841
Epidemic	0.8813	15621.2	669193	860475

CacheDTN has low hit rate, median cost, and high average delay. This is because though replicas are created, requests wait for files passively on their originators, leading to a long delay. Also, the replication of files to network centers incurs a high cost. PodNet has low hit rate for the same reason as CacheDTN. However, since the replicas on each node are more catered for the interests of itself and nodes it has met, PodNet has slightly higher hit rate than CacheDTN. Moreover, PodNet has the lowest cost because nodes in it only replica files they are interested in. PDI+DIS generates the lowest hit rate at relatively high total cost and low average delay. The low hit rate is caused by the poor mobility resilience of route tables. As a result, only partial requests are resolved quickly in the local broadcasting. Others passively wait for file holders or updated routes and usually cannot be resolved timely. Then, since we only count the average delay of successful requests, PDI+DIS has the lowest average delay.

Table 4.4: Memory usage in the experiments on GENI

Metric	SPOON	MOPS	CacheDTN	PodNet	PDI+DIS	Epid.
Request	36.8	44.1	48.4	45.3	13.1	1998
Table	10.4	16.9	50	50	15.6	0

We also evaluated the performance of each method in memory utilization in terms of the average number of requests in the buffer (Request) and the average size of a content/neighbor table (Table). The results are shown in Table 4.4. For the average number of buffered requests, we find that PDI+DIS<SPOON< MOPS<PodNet<CacheDTN<Epidemic. In Epidemic, nodes buffer the most requests since it tries to replicate each request to all nodes in the system. CacheDTN and PodNet have a lot of requests in memory since they does not actively search for the requested files. Both SPOON and MOPS keep one copy of each request during the searching process. However, since SPOON completes file request more quickly than MOPS (as shown in Table 4.3), it buffers



(a) Hit rate.

(b) Average delay.

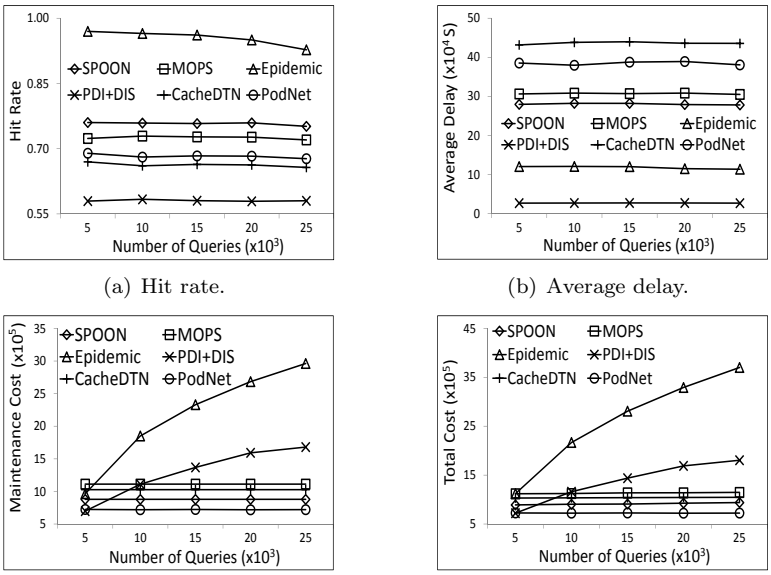
(c) Maintenance cost.

(d) Total cost.

Figure 4.4: Performance in the event-driven experiments with Haggle trace.

fewer quires in memory than MOPS. PDI+DIS has the fewest number of requests on nodes because local broadcasting just forwards requests without buffering.

Considering that each entry in the content table has roughly the same size, we used the number of entries in a table to represent memory usage. The results in Table 4.4 show that Epidemic<SPOON<MOPS< PDI+DIS<CacheDTN=PodNet. Clearly, Epidemic does not need memory on content table. SPOON stores the second fewest content synopses because most nodes only store the information of the same community members. In MOPS, brokers store content synopses of all nodes in their communities and consume a large amount of memory. Therefore, MOPS produces high average number of stored content synopses. PDI+DIS stores a alrge amount of content synopses because each node collects content synopses from all nodes it has met and from all received reply messages. CacheDTN and PodNet have the highest number of entries in the content table since we doubled the memory for replicas. In summary, the results in Table 4.3 and Table 4.4 show that SPOON is superior over other methods in terms of hit rate, average delay, total cost, and memory efficiency.



(a) Hit rate.

(b) Average delay.

(c) Maintenance cost.

(d) Total cost.

Figure 4.5: Performance in the event-driven experiments with MIT Reality trace.

4.5.3 Event-driven Experiments with Real Traces

In this experiments, we varied the total number of requests from 5000 to 25000 to show the scalability of each method in terms of the amount of requests.

4.5.3.1 Hit Rate

Figures 4.4(a) and 4.5(a) show the hit rate of each method in the experiments with the Haggle trace and the MIT Reality trace, respectively. We find that with both traces, Epidemic can resolve almost all requests, while PDI+DIS can only complete about 60% of requests. The hit rates of SPOON, MOPS, PodNet, and CacheDTN reach about 75%, 70%, 68%, 67%, respectively. Epidemic has the highest hit rate because of its broadcasting nature. In SPOON, coordinators and ambassadors facilitate intra- and inter- community searching, while the IRA actively forwarded to the node with a high probability of meeting the destination. MOPS only relies on the encountering of mobile brokers for file searching. This probability is lower than that of SPOON, resulting in a lower hit rate. PodNet and CacheDTN lack active request forwarding, leading to median hit rates. However, replicas on each node are more catered to the interests of nodes it can meet in PodNet, while CacheDTN just caches files on network center. Therefore, PodNet has slightly higher hit rate than CacheDTN. In PDI+DIS, many routes in the content table expire quickly due to node

mobility. As a result, most successful requests are resolved through the 3-hop broadcast. Others have to passively wait for file holders or updated routes. Therefore, many requests cannot be resolved, leading to a low hit rate.

4.5.3.2 Average Delay

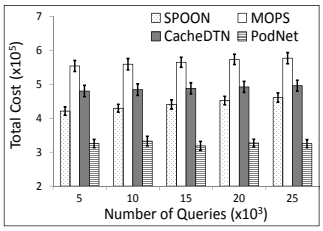
Figures 4.4(b) and 4.5(b) show the average delays of the six methods in the tests with the Haggle trace and the MIT Reality trace, respectively. The delays follow PDI+DIS< Epidemic< SPOON<MOPS<PodNet<CacheDTN.

Recall that we only measure the delay of successful requests. In PDI+DIS, most successful requests are resolved in the initial 3-hop broadcasting stage. Therefore, it generates the least average delay. In Epidemic, requests are rapidly distributed to nodes at the cost of multiple copies. As a result, requests can reach their destinations quickly. MOPS exhibits a large delay because requests in it usually have to wait for a long time for brokers or same-community file holders. In contrast, SPOON always tries to find an optimal neighbor to send a request to the file holder with the interest-oriented routing algorithm. In addition, the designation of ambassadors in SPOON increases the possibility of relaying requests to foreign communities. As a result, SPOON has lower average request delay than MOPS. PodNet and CacheDTN generate high average delay because requests only wait for file holders passively on their originators. However, since PodNet create replicas that are more likely to be encountered by nodes that are interested in them, it has lower average delay than CacheDTN.

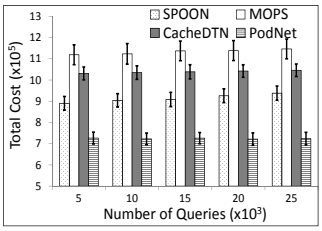
4.5.3.3 Cost

Figures 4.4(c) and 4.5(c) plot the maintenance costs of the six methods in the experiments with the Haggle trace and the MIT Reality trace, respectively. We see that when the total number of requests is small, the six methods all have low maintenance cost. When the total number of requests is larger than 10000, the maintenance costs generally follow: Epidemic>PDI+DIS>MOPS> CacheDTN> SPOON>PodNet.

In PodNet, nodes only replica interested files when meet other nodes, leading to the least maintenance cost. In SPOON, nodes exchange node vectors for the update of history vector. Nodes also report its contents to coordinators for file indexing. In MOPS, brokers exchange the contents of all nodes from their home communities when meeting each other. Therefore, MOPS produces slightly



(a) Haggle trace.



(b) MIT Reality trace.

Figure 4.6: Total costs with confidence intervals.

higher cost than SPOON. The active replication of files to network centers in CacheDTN leads to a high cost. PDI+DIS needs to build content tables through reply messages and disseminated requests, so it has higher maintenance cost than above four methods when the number of requests becomes large. In Epidemic, two nodes need to inform each other requests already on them, which causes a lot of information exchange and leads to the highest maintenance cost.

We see that when the number of requests increases, the maintenance costs of SPOON, PodNet, CacheDTN, and MOPS remain stable while those of Epidemic and PDI+DIS increase quickly. This is because the maintenance costs of the former four methods are determined by the information/replication exchanges among nodes and are irrelevant with the number of requests, and those of Epidemic and PDI+DIS are related to the total number of requests. Such results prove the scalability of SPOON, MOPS, CacheDTN, and PodNet in request amount.

Figures 4.4(d) and 4.5(d) show the total cost of each method in the experiments with the Haggle trace and the MIT Reality trace, respectively. In the two figures, the results of MOPS, CacheDTN, SPOON, and PodNet are shown to be very close. We then plot the total costs of the four methods with 95% confidence interval in Figure 4.6(a) and Figure 4.6(b) for better demonstration. Note we did not show the confidence interval of other measurements because they have clear difference. We find that the total costs follow Epidemic>PDI+DIS>MOPS>CacheDTN > SPOON>PodNet, which is the same as Figures 4.4(c) and 4.5(c). Such a result means that the maintenance cost is the majority part of the total cost. With above results, we conclude that SPOON has the highest overall file searching efficiency in terms of hit rate, delay, and cost.

4.5.4 Evaluation of the Enhancement Strategies

4.5.4.1 Multi-copy forwarding and Prefetching

We first evaluated the effect of the multi-copy forwarding and the intelligent file prefetching. We let "Multi-copy forwarding" and "Prefetching" denote the SPOON with the corresponding improvement strategies, respectively, and compare them with the "Original" SPOON. In Multi-copy forwarding, we let each request originator distribute two copies of its request. In Prefetching, we let each ambassador store top ten most popular files. We varied the number of file requests from 5000 to 25000. The test results are shown in Tables 4.5 and 4.6.

We find that the multi-copy forwarding strategy with only two copies enhances the hit rate greatly in the experiments with both traces. This is because when each request has two copies in the system, its probability of encountering the node containing the requested file increases. Such a result shows the effectiveness of the multi-forwarding strategy. We also see from the two tables that the file prefetching strategy slightly improves the hit rate with both the two traces. This is because 1) we only configure two ambassadors per community, and 2) the prefetched files only satisfy a small amount of requests since most requests are for contents in local community. The improvement on the hit rate still demonstrates the effectiveness of the file prefetching strategy and the improvement would be greater with more ambassadors and greatly varied file popularity.

Table 4.5: Hit rate improvement with the Haggle trace.

# of packets	Original	Prefetching	Multi-copy forwarding
5000	0.75137	0.75313	0.779412
10000	0.75215	0.75633	0.780135
15000	0.73831	0.74274	0.778912
20000	0.74928	0.75242	0.774321
25000	0.74731	0.75201	0.779415

Table 4.6: Hit rate improvement with the MIT Reality trace.

# of Packets	Original	Prefetching	Multi-copy forwarding
5000	0.761371	0.7671675	0.780413
10000	0.759941	0.762841	0.770838
15000	0.760135	0.763762	0.772843
20000	0.756418	0.759957	0.773901
25000	0.751835	0.754837	0.771963

4.5.4.2 Request-Completion and Loop-Prevention

We name SPOON without request-completion as "SPOON-OR", SPOON with the request-completion only as "SPOON-QC", and SPOON with both request-completion and loop-prevention

as "SPOON-QCLP". We set the number of requests to 15000. In SPOON-OR, requests do not stop searching until TTL expiration. We also set the maximal number of files each request can retrieve, S_{max}, to 2. To enable a request to find two files, we purposely let each file have two copies in the system. In order to alleviate the influence of the TTL on the evaluation of the request-completion, we enlarge the TTL to the entire length of the used traces. The test results are shown in Table 4.7.

We find from the table that SPOON-QCLP has slightly higher hit rate than SPOON-QR and SPOON-QC. This is because the loop-prevention avoids forwarding a request to the same file holder repeatedly, thereby utilizing forwarding opportunities more efficiently. SPOON-QC has slightly lower hit rate than SPOON-OR because it stops file searching when S_{max} files are fetched. We also see that the number of request forwarding operations follow SPOON-OR>SPOON-QC>SPOON-QCLP. This is because SPOON-OR does not stop file searching until the TTL is expired. SPOON-QC reduces the cost as it stops file searching after the specified number of files are located. In SPOON-QCLP, the loop-prevention avoids redundant forwarding to the same node, leading to smaller number of forwarding operations and more efficient file searching.

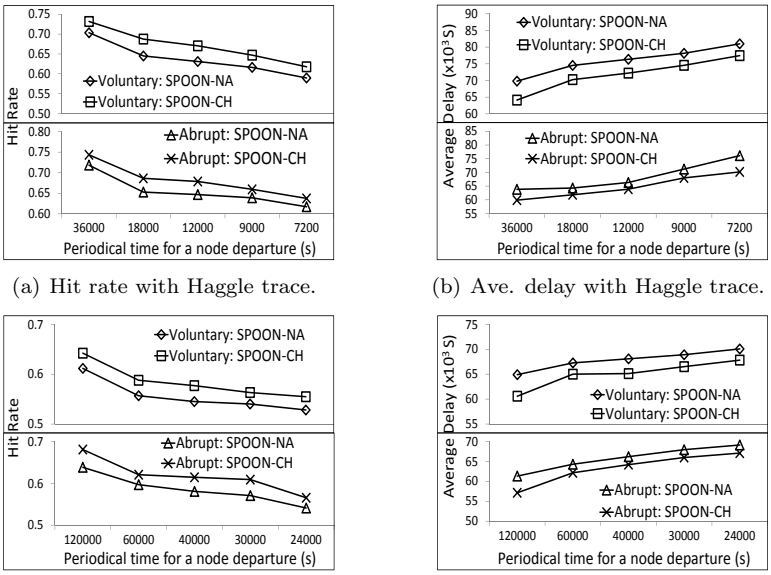
Table 4.7: Effect of request-completion strategy.

Trace	SPOON-OR	SPOON-QC	SPOON-QCLP
Hit Rate			
Haggle	0.8741	0.8701	0.8813
MIT Reality	0.9091	0.9012	0.9406
Number of request forwarding operations			
Haggle	569841	530516	304953
MIT Reality	721852	609863	249132

4.5.4.3 Node Churn Consideration

We name SPOON with and without a strategy to handle node churn as "SPOON-CH" and "SPOON-NA", respectively. The total number of requests was set to 15000. The period for beacon message was set to 100s and 1000s with the Haggle trace and the MIT Reality trace, respectively. In the test, N_L nodes leave the system evenly during the first 1/2 and 1/4 of the Haggle trace and the MIT Reality trace, respectively. N_L was varied from 5 to 25. We name the node that contains a file matching a request as the *primary matching node* for the request. In order to demonstrate the performance of SPOON in node churn, for each requested file, we purposely created a file that has 70% similarity with it in a non-leaving node in the same community with the primary matching node. We name this node as the *secondary matching node*.

The test results of voluntary and abrupt normal nodes departure are shown in Figure 4.7.



(a) Hit rate with Haggle trace.

(b) Ave. delay with Haggle trace.

(c) Hit rate with MIT trace.

(d) Ave. delay with MIT trace.

Figure 4.7: Performance with voluntary and abrupt node departures.

We see that in all cases, when node churn consideration is applied, the hit rate is increased and the average delay is decreased. This is because with the node churn consideration, requests failing to find their primary matching nodes (i.e., have left the system) are further forwarded to their secondary matching nodes while when there is no node churn consideration, these requests just wait on coordinators for the primary matching node, leading to a low hit rate and a high average delay. We also observe that when the number of leaving nodes increases, the hit rate decreases and the average delay increases. This is because leaving nodes can no longer relay requests or departure notification/detection messages, leading to lower hit rate and higher average delay.

Table 4.8: Effect of the detection of coordinator departures.

Trace	w/o churn consideration	w/ churn consideration-Ab	w/ churn consideration-Vo
Haggle	0.650643	0.684512	0.729942
MIT Reality	0.641053	0.677841	0.746841

We also tested the scenario in which only coordinator nodes leave the system. Since the total number of coordinators is limited, we only randomly chose two coordinators to leave the system during the test. We name the scenarios when coordinators depart abruptly and voluntarily as "w/ churn consideration-Ab" and "w/ churn consideration-Vo", respectively. The test results are shown in Table 4.8. We find that when coordinators leave the system, the hit rate of SPOON with node

churn consideration is much higher than that without node churn consideration. This is because the coordinator is critical in both intra- and inter- file searching in SPOON. Without node churn consideration, requests just wait for coordinators until their TTL expire if they need to be forwarded to coordinators, leading to a low hit rate and a high average delay. Above results show that SPOON's strategies for node churn consideration can improve the system performance at a low cost.

4.6 Summary

In this chapter, we propose a Social network based P2P cOntent file sharing system in mobile Opportunistic Networks (SPOON). SPOON considers both node interest and contact frequency for efficient file sharing. We introduce four main components of SPOON: *Interest extraction* identifies nodes' interests; *Community construction* builds common-interest nodes with frequent contacts into communities. The *node role assignment* component exploits nodes with tight connection with community members for intra-community file searching and highly mobile nodes that visit external communities frequently for inter-community file searching; The *interest-oriented file searching scheme* selects forwarding nodes for requests based on interest similarities. SPOON also incorporates additional strategies for file prefetching, request-completion and loop-prevention, and node churn consideration to further enhance file searching efficiency. The tests on the GENI Orbit platform and the trace-driven experiments prove the efficiency of SPOON.

Chapter 5

Utilizing Distributed Social Map for Lightweight Packet Routing among Nodes in MONs

In previous two chapters, we have investigated how to realize efficient file sharing in MONs. In this chapter, we explore how to realize efficient packet routing among nodes in MONs. Considering the social network property that the people a person frequently meets are usually stable and these people play an important role in forwarding packets for the person [17], we propose a social map based MON routing algorithm, denoted by SMART. The social map on each node records its surrounding social network in MONs and is constructed by learning each encountered node's most frequently met nodes (i.e., stable friends). To construct the social map, two encountered nodes only need to exchange the information of their most frequently encountered nodes. Figure 5.1 shows an example of social map on Bob. Each link in the social map is associated with a weight based on the encountering frequency and social closeness of the two connected nodes. The weight is used to deduce the delivery abilities among nodes. Therefore, each node only needs to check its own social map to make forwarding decisions, i.e., which packets should be forwarded to the other node. For example, when Bob meets Allen, without querying Allen's probabilities to meet other nodes, he would know that packets for Emma, Frank or Glair should be forwarded to Allen. We can see that the social map is not limited to one or two hops and reflects possible long relay paths to provide

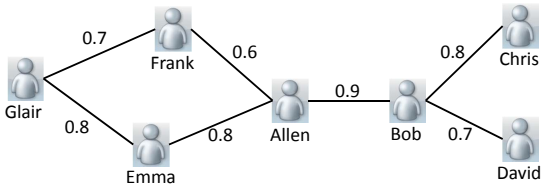


Figure 5.1: The social map of Bob.

better forwarder selection. SMART does not require social maps to be identical in all nodes or to be complete (i.e, including all nodes), which makes the social map construction simple and suitable for distributed MONs. Further, the stability of most frequently encountered nodes means that no frequent social map update is needed, which reduces resource consumption. As a result, the proposed SMART algorithm can realize efficient and effective packet routing among nodes in MONs.

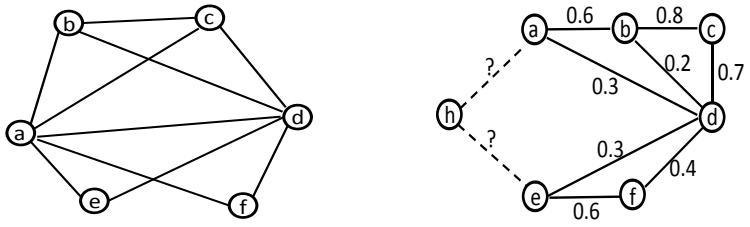
In the following, we first explain the benefits of social map on routing efficiency in Section 5.1. We then present how the social map is constructed in Section 5.2. After this, we introduce the detailed social map based routing algorithm in Section 5.3. We also provide some discussions on the scalability and security of the proposed SMART algorithm in Section 5.4. The real-trace driven performance evaluation is given in Section 5.5. Finally, Section 5.6 summarizes this chapter

5.1 The Benefits of Social Map on Routing Efficiency

We first discuss the benefits of social map from the perspective of routing efficiency with a simple scenario, shown in Figure 5.2(a). We denote the meeting probability and delivery ability between two nodes as P_{ij} and D_{ij} $(i, j \in \{a, b, c, d, e, f\})$, respectively. The former is the probability of delivering a message to another node directly upon their encountering. The latter refers to the probability of delivering a message to another node through either direct forwarding or indirect relay. We assume d is the destination node.

5.1.1 Drawback of Previous Methods

In routing algorithms that use delivery ability, when a meets b, it updates its delivery ability to d (D_{ad}) by considering the relay through b. In PROPHET [72], $D_{ad} = D_{ad}+(1-D_{ad})*P_{ab}*P_{bd}*\beta$, in which $\beta \in [0, 1]$ is a scaling constant. Such updates only consider two-hop relay delivery ability (i.e., $a \rightarrow b \rightarrow d$), which has limited view on forwarder selection and may miss a forwarder on a



(a) A small network.

(b) An example on route selection.

Figure 5.2: A network scenario to show the benefits of social map.

faster but longer path.

One may claim that using transitive probability calculation can provide much wider view. That is, using b's delivery ability to d (D_{bd}) to update D_{ad}: $D_{ad} = D_{ad} + (1 - D_{ad}) * P_{ab} * D_{bd} * \beta$. Since D_{bd} is already calculated based on all routes from b to d (e.g., $b \rightarrow c \rightarrow d$), the updated D_{ad} can reflect routes more than two hops (e.g., $a \rightarrow b \rightarrow c \rightarrow d$). However, this may lead to delivery ability calculated for a routing path with loops (e.g., $a \rightarrow b \rightarrow a \rightarrow e \rightarrow c \rightarrow d$). We see from the equation that D_{ad} is updated by D_{bd}. But similarly, D_{bd} may be updated by D_{ad} previously, which means D_{bd} has already considered relaying through a. Therefore, by updating with D_{bd}, D_{ad} integrates the relay though itself. In other words, D_{bd} and D_{ad} may boost each other repeatedly, leading to inaccurate delivery ability.

We confirm this problem with real traces from the MIT Reality project [34] and the Haggle project [18]. The former was obtained from students and staffs on the MIT campus, while the latter was collected from 98 scholars attending Infocom'06. Both traces include encountering records among people. Table 5.1 shows the summary of the two traces.

Table 5.1: Characteristics of mobility traces.

	MIT Reality	Haggle
# Nodes	94	98
Location	Campus	Conference
Duration	30 days	4 days
# Encountering	137936	74224
# Encountering per day	4597	18556

We set β to 0.5 and measured the delivery abilities of all nodes to a randomly selected node using P_{bd} and D_{bd}, respectively. The average delivery abilities of all nodes are 0.43 and 0.70 in the MIT Reality trace, respectively, and are 0.2 and 0.42 in the Haggle trace, respectively. We see that by replacing P_{bd} with D_{bd}, the delivery ability is exaggerated greatly, thereby cannot provide

accurate forwarder selection guidance. Thus, it is not feasible to use the transitive probability to enlarge the view during forwarder selection. We then propose social map for this purpose with a controllable cost.

5.1.2 Benefits of Social Map

The social map on a node provides a much broader view naturally. A node can discover routes to the destination with any lengths, thus providing more accurate forwarder selection. Figure 5.2(b) gives a simple example, in which the number on each link represents the meeting probability between the two connected nodes. Suppose each node has learnt their meeting probabilities. Node h needs to select a node from a and e as the next hop for a packet towards node d. Without social map, PROPHET cannot consider relay routes that are more than two hops. Then, since D_{ad} is $0.3 + 0.6 * 0.2 = 0.42$ and D_{ed} is $0.3 + 0.6 * 0.4 = 0.54$, node e is a better forwarder than node a. With social map, we can consider longer routes (i.e., $a \rightarrow b \rightarrow c \rightarrow d$) for D_{ad}: $0.3 + 0.6 * 0.2 + 0.6 * 0.8 * 0.7 = 0.756$, which is larger than D_{ed}. Then, node h can select the correct forwarder (i.e., a). There are already several ways to compute the weight between two nodes in a graph when the weight of each edge is known [32, 70, 79]. These methods require the weight of every edge in the network and much calculation for each source-destination pair, which cannot be satisfied in MONs, i.e., a node cannot know all link weights and the routing process needs to calculate the weights between many pairs of nodes. Therefore, we adopt a different way to calculate the delivery ability between two nodes on the social map, as explained in Section 5.3.1.

The social map provides benefits on routing, which are actually resulted from the cost of maintaining more information (i.e., top L friends) on each node. The social map only contains a node's major relationship, which is stable and requires less frequent updates. Therefore, it actually provides an acceptable balance on cost and routing performance.

5.2 Social Map Construction

Ideally, the social map should include all nodes in the system. However, this would consume extensive resources for information exchange and storage. Also, the social map structure should be stable to reduce the necessity of timely update, which is hard to realize in MONs. Moreover, the social link weight should be able to reflect the delivery possibility between connected nodes for

efficient routing. These problems pose two challenges: *i) how to build stable social maps with a low maintenance cost? ii) How to define the link weight that can accurately reflect delivery ability?* We introduce our solutions to these challenges in below.

5.2.1 Lightweight Social Map Construction

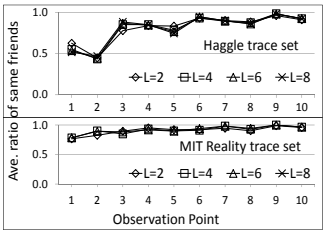
In a social network, a person usually meets his/her major social relations frequently, who play a more important role in his message forwarding [17]. For example, we meet the same colleagues, friends, and family members daily. Inspired by this, we only keep the nodes a node has met and their top L most frequently encountered nodes (called **top L friends**) in the node's social map. L can be a fixed value or the number of encountered nodes whose meeting frequencies with the node are higher than a predefined threshold. Due to the stability of a node's top L friends, the social map requires low cost for the structure maintenance and update. In below, we first show the stability of top L friends and then introduce the social map construction process, the coverage of the resulted social map, the ways to determine the value of L, and the resulted cost saving.

5.2.1.1 Stability of Top L Friends

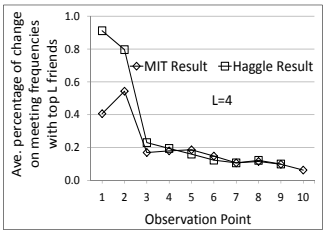
In order to verify the stability of a node's top L friends and the frequencies of meeting them, we analyzed the MIT Reality project [34] trace and the Haggle project [18] trace. We see from the summary in Table 5.1 that nodes move actively in the two traces.

We set ten observation time points evenly in the two traces. At each observation point, we generated the top L friend lists of each node and calculated the ratio of the same top L friends as $\frac{|F_i \cap F_{i+1}|}{|F_i|}$, in which F_i and F_{i+1} denote the set of top L friends at observation point i and $i+1$, respectively. In order to get F_i, SMART counts the encounters from the start time to observation point i. We measured the ratio when L equal to 2, 4, 6, and 8. The average ratios of all nodes are shown in Figure 5.3(a). We can see that after the initial two observation points, the average ratio remains very high (around 90%). Note that the length between two observation points is quite long in the experiment. This result confirms that a node's most frequently met nodes are very stable.

We further measured the variance of each node's meeting frequencies with its top L friends over time. We first ran the trace and selected the top L most frequently met nodes as the top L friends in this measurement. The frequency change of a node to its friend is measured by $\frac{|f_{i+1}-f_i|}{f_i}$, in which f_{i+1} and f_i denote the meeting frequency with the friend at observation point i and $i+1$,



(a) Evolution of top L friends.



(b) Evolution of the change of meeting frequency with top L (4) friends.

Figure 5.3: Evolution on the change of friend list and meeting frequency.

respectively. Figure 5.3(b) shows the average of all nodes' frequency changes when L equals 4. We see that in both traces, the frequency change is large only at the beginning and decreases to less than 20% after the first two observation points and finally reaches about 10%. This result verifies that a node's meeting frequencies with its top L friends are also relatively stable.

Though the above results are obtained from only two traces with around 100 nodes, they can represent human-based MONs to a certain extent. Firstly, the two traces represent two typical human-based MON scenarios (campus and conference site). Secondly, such results match our daily experiences that we often meet the same group of people regularly, e.g., colleagues, family members, and friends.

5.2.1.2 Social Map Construction Process

We first introduce a concept of *friendship rank*. We divide high meeting frequencies in the system to a number of ranges and assign a rank to each top L friend based on meeting frequencies. Each node then matches its meeting frequencies with other nodes to these ranges to determine its friendship ranks with other nodes. The ranges can be determined using both centralized and distributed methods. In the centralized method, the system administrator pre-determines the ranges based on the application scenario. To obtain the meeting frequencies among nodes in the system, a collector can be placed in a popular place to collect the meeting frequencies from nodes. In the distributed method, nodes exchange their meeting frequencies in a gossip manner. Then, when the designated node has collected the meeting frequencies of most nodes in the system, it decides the meeting frequency ranges and informs all other nodes through broadcasting. Therefore, this method has a high overhead. The system owner can select a suitable method based on the application requirement. As observed from Figure 5.3(b), a node's meeting frequencies with its top L friends

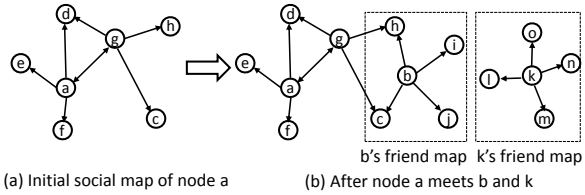


Figure 5.4: Social map update process.

are relatively stable. Therefore, the ranks of a node's top L friends are relatively stable. The reason we use the rank instead of meeting frequency is that it can reduce the cost in social map updates caused by the fluctuation of meeting frequencies.

We define each node's top L friends and their friendship ranks as its *friend map*. When two nodes meet, they exchange and update their friend maps. Each node maintains a *social table* that records friend maps of all nodes it has met, as shown in Table 5.2. A node's social map is constructed by connecting all nodes in its social table, and each node only appears once in the map. A directional link from node i to node j means node j is in the top L friend list of node i. Figure 5.4(a) shows an example of the social map of node a with L=4.

Table 5.2: Social table

Node	Top L friends	Friendship ranks
a	f, e, d, g	1, 2, 3, 4
g	d, a, c, h	3, 4 ,4, 5
b	h, c a, j	2, 2, 3, 5
k	i, o, m, n	1, 1, 3, 4
...	...	...

The social map on a node, say node a, is updated after each encountering with another node rather than at a specific time spot. Specifically, when node a meets node b and receives its friend map, if b is already in a's social map, node a updates b's L connected nodes in the social map accordingly. Otherwise, node a integrates b's friend map into its social map. Figure 5.4 demonstrates the update process of node a's social map after it meets node b and k. When a meets b, it learns b's top L friends (c, h, i, and j). As h and c are already in the map, node a only adds b, i, and j to its social map. There is no partition in the network. Later, node a meets node k, whose friend map contains l, m, n, and o. Since none of them are in a's social map, a partition is created after they are added into a's social map, as shown in Figure 5.4(b). In this case, we still regard it as part of the social map because 1) the partition still shows some information of the network (i.e., o, l, m, n are good relays for node k), and 2) nodes that can connect partitions may be encountered

and inserted into the social map later. Note that though the social map on each node is updated upon each encountering, this process does not consume significant resources since a node's top L friends usually are stable, as shown in Figure 5.3(a).

Above algorithm finally generates social maps that are not identical in all nodes and may not include all nodes. This is similar to our daily lives that each person has his/her own knowledge of the social structure. We will see that the routing efficiency can still be ensured later in Section 5.5.

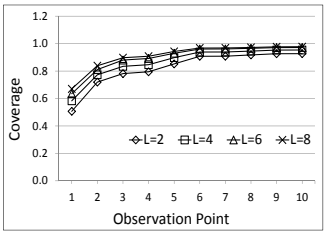
5.2.1.3 Social Map Coverage

We define the *coverage* of the social map as the number of nodes in the map divided by the total number of nodes in the system. We then measured the average coverage of the social map on each node at 10 evenly distributed observation points with different L values. The results with the Haggle trace and the MIT Reality trace are shown in Figures 5.5(a) and 5.5(b), respectively. We see that even when $L = 2$, after a short period of time, the social map on each node can cover over 80% of nodes in the Haggle trace and 60% of nodes in the MIT trace. When $L = 8$, the social map coverage in the two traces increase to 90% and 80%, respectively. Such results demonstrate the feasibility of using social map to provide routing guidance.

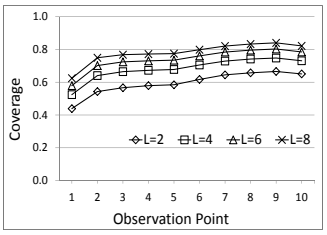
5.2.1.4 Determining the Value of L

Clearly, the value of L affects the social map on each node. When L increases, the social map contains more information and can provide better routing guidance. However, larger L also consumes more communication and storage resources. Therefore, we need to balance the value of L and the cost. As aforementioned, the goal of the social map is to reflect stable social relationships. Then, we can determine L based on the number of stable friends of each node. The stable friend list of a node contains the frequently met nodes whose meeting frequencies with the node are larger than half of that of the most frequently met node. For example, suppose node j is the most frequently met node of node i, and the meeting frequency between them is f_{ij}. Then, nodes whose meeting frequencies with node i are larger than $f_{ij}/2$ are stable friends of node i. The determination of L can be realized in both centralized and distributed manner.

In the centralized method, we first let node run for a period of time so that all nodes can collect enough amount of encountering records. Then, we periodically calculate L as the average number of stable friends each node has. The updated L is sent to each node whenever its connection



(a) Haggle trace.



(b) MIT Reality trace.

Figure 5.5: Social map coverage with different Ls.

to the central server is established. In the distributed method, considering different nodes have different social relationships and consequently, different numbers of stable friends, we do not require the L to be identical among all nodes. Each node directly uses the size of its stable friend list as L.

Both the two methods have advantages and disadvantages. In the centralized method, with the determined L, mobile nodes can save the cost on exchanging L value. However, it suffers from the problem that it cannot adjust L timely and needs a central/super node to collect necessary information. In the distributed method, the social maps get updated timely when the stable friends of a node change, providing more accurate routing information for packets. However, the drawback of the distributed method is that it is hard to control the size of social map since individual nodes decide their own Ls.

Some may question that the social map constructed with limited L may fail to reflect some forwarding opportunities, especially for active nodes, since each node can only have most L friends in the social map. We argue that it does not sacrifice the routing performance because 1) the top L friends reflect the major social relationships of each node, which usually take the major roles in message forwarding, and 2) active nodes would appear as the top L friends of more nodes, thus having more links in the social map. Therefore, the specified L does not compromise routing performance, which is verified in our analysis in Section 5.3.3 and experiments in Section 5.5.

5.2.1.5 Cost Saving Resulted from Social Map

We assume the average value of L on each node is K no matter different nodes have the same L or different Ls. Then, on average, two encountered nodes only exchange their top K friends and associated friendship ranks for social map construction. Suppose the size of the information of one friend is T bytes, the total amount of data exchanged is about $2TK$ bytes. In previous methods, two encountered nodes exchange their delivery abilities to the destinations of all packets

to make forwarding decision. The size of each delivery ability can be regarded as T bytes too since it also represents the information of a node. We assume packets on each node have N different destinations on average. Then, the total amount of data exchange is about $2TN$ bytes. As a result, the total cost saving is $2T(N-K)M$, where M is the total number of encounters. Recall that N is bounded by the total number of nodes in the system. Also, the higher the packet generation rate is, the larger N tends to be (suppose packet destinations are evenly distributed). On the other hand, K usually is small ($K < 10$). Therefore, the social map can reduce the information exchange cost in most cases (i.e., when the packet generation rate ensures $N > K$). Furthermore, the cost saving is more valuable when the packet generation rate is high because more packets require more storage/forwarding resources in the resource-limited MONs.

5.2.2 Social Link Weight Calculation

We assign a weight to the link connecting two nodes, say node i and node j, in a social map to represent how fast a packet can be forwarded between them. We consider two factors in this process: the meeting frequency and the social closeness between the two nodes, which are reflected by shared top L friends [30]. We consider social closeness for weight calculation because people with close relationship are likely to share the same group of friends [30]. A shared top L friend of two nodes is a good relay to forward messages between them since both of them meet the friend frequently. Then, the resultant link weight can more accurately reflect the message delivery ability between the two connected nodes.

We call the link path directly connecting two nodes as 1-hop route and the link path connecting two nodes through one shared top L friend as 2-hop route. The weight of a l-hop ($l = 1$ *or* 2) route between node a and node b, denoted by w_{ab}, is defined as 1 over the sum of the friendship rank of each link in the route:

$$w_{ab} = 1/\sum_{k=0}^{l-1} r_k \tag{5.1}$$

where r_k denotes the rank of the k^{th} link. For two nodes, the weight of one-hop route reflects their meeting frequency while the two-hop routes show the social closeness.

Then, the weight for a link in the social map connecting node a and node b, denoted by

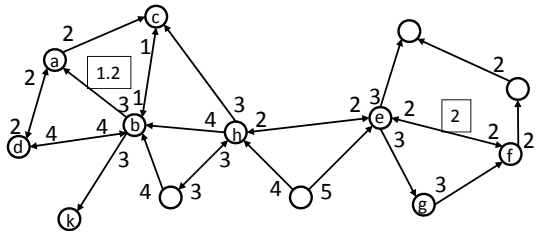


Figure 5.6: Part of node h's social map.

W_{ab}, integrates all one-hop and two-hop routes between them.

$$W_{ab} = 1/\sum_{s=0}^{m-1} w_{ab_s}, \tag{5.2}$$

where m is the total number of routes and w_{ab_s} is the weight of the s^{th} route. With this design, the smaller W_{ab}, the higher forwarding probability the two nodes have.

Figure 5.6 shows an example of part of the social map created on node h, in which L equals 4. We briefly introduce how to calculate the link weight W_{ab} and W_{ef}. There are three routes between a and b: one one-hop route $(b-a)$ and two two-hop routes through shared top L friend d and c ($a-c-b$ and $a-d-b$). Based on Equation 5.1, these routes' weights are 1/3, 1/3, 1/6, respectively. Based on Equation 5.2, $W_{ab} = 1/(\frac{1}{3}+\frac{1}{3}+\frac{1}{6}) = 1.2$. As for e and f, they have only one route: $e-f$. Route $e-g-f$ is not a two-hop route since g is not a shared top L friend, as it only exists in the top L friend list of e. Therefore, $W_{ef} = 1/\frac{1}{2} = 2$.

5.2.2.1 Understanding the Social Link Weight

The friendship rank can represent the expected inter-meeting time between the two connected nodes, i.e., the expected delay for a packet to be relayed between the two nodes, as explained in the definition of friendship rank. As a result, the sum of each link's friendship rank in a route (i.e., $\sum_{k=0}^{l-1} r_k$ in Equation (5.1)) represents the expected delay of replaying a packet through the route. Then, the weight of a route, calculated in Equation (5.1), actually represents the "normalized throughput" of the route. Note that in above discussion, we call "normalized throughput" because we assume each node's memory can hold the same amount of packets. This can be adapted to practical scenarios by deciding the average amount of memory on each node.

The overall "normalized throughput" between two node, say a and b, is the sum of each route's "normalized throughput" : $\sum_{s=0}^{m-1} w_{ab_s}$. Then, the weight of the social link connecting

two nodes, which is calculated as 1 over the "overall throughput" (Equation (5.2)), represents the expected delay to relay a packet from one node to the other node through the possible route considered in our design.

5.3 Social Map Based Routing Algorithm

In this section, we first introduce how we utilize the social link weight and then present the detailed routing process.

5.3.1 Deciding Delivery Ability

In MON routing, a packet is always forwarded to the candidate forwarder that has the highest ability to deliver it to its destination. Then, how to decide the delivery ability with the social link weight?

For two nodes in the social map, there are multiple paths connecting them. We define the weight of a path on the social map as the sum of the weights of its social links. Since the weight of a social link denotes the expected delay to forward a packet between the two connected nodes, the weight of a path represents the expected delay to pass a packet through the path. Then, the weight of the minimal weight path from the holder to the destination of a packet represents the minimal expected time needed to deliver the packet. Since delay is the crucial factor in MON routing, we take the weight of the minimal weight path to represent the delivery ability between the two nodes. This also matches the general MON routing process, which forwards packets hop by hop and aims to maximally reduce the expected delay in each forwarding hop.

5.3.2 Routing Process

With above analysis, the guideline of our routing algorithm is to always forward a packet to a node whose minimal-weight path to the destination node has smaller weight, i.e., gradually and maximally reducing the minimal expected delay. In detail, suppose node a needs to decide whether node b is a better forwarder for one of its packet. Node a first checks whether both a and b are disconnected to the destination node in the social map. If yes, we use a backup metric to decide the forwarder, as introduced later. Otherwise, node a uses the Dijkstra [31] algorithm to find the minimal-weight paths from the destination node to a and b. Note if a or b is disconnected to the

destination node, the weight of its minimal-weight path to the destination node is the maximal value. If node b's minimal-weight path has smaller weight than that of node a, node a would forward the packet to node b.

Besides, there are some issues that need to be addressed, such as incomplete social map, loop prevention, and packet replacement strategy. We first discuss approaches to solve these problems and then summarize the routing algorithm.

a) Incomplete Social Map: As stated previously, the social map on each node may only cover part of the entire network. Though we find in Section III-B1c that the social map has high coverage, it is possible that when a node meets another node, the destination node of a packet is disconnected from the two nodes in the social map (i.e., no path can be found for both nodes). In this case, we rank a node's suitability of forwarding a packet by its active degree, which is measured by the number of links associated with the node in the social map. The more links connecting to a node, the more active it is. Then, the packet is forwarded to the node with higher active degree. This is inspired by the social network property that an active person can meet more people and thus has a higher probability of meeting the destination [17].

b) Loop Prevention: Since each node maintains its social map independently, forwarding loops may happen in the system. For example, two nodes may both believe that the other side is a better forwarder for a packet and forward the packet back and forth repeatedly. To prevent such a loop, we require each packet to record the IDs of all nodes that it has been forwarded to. Then, the loop can be avoided by simply forbidding a node to forward a packet to a node that it has visited before.

c) Packet Replacement: It is possible that a node's storage is full when a packet arrives. In this case, SMART simply drops the packet that has lived for the longest period of time.

We then summarize the routing algorithm in SMART as below, with its pseudo-code shown in Algorithm 1.

(1) When two nodes meet with each other, they first exchange their friend maps, which are then used to update their social maps (line 2-4). After this, each node processes its packets sequentially (line 7).

(2) For the current packet, the node first checks if it has been forwarded to the other node before. If not, it proceeds to step (3). Otherwise, it goes to step (5) (line 9).

(3) If the destination node connects to at least one of the two nodes in the social map, the node checks whether the other node's minimal-weight path to the destination node has lower weight with the Dijkstra algorithm. Otherwise, the node checks whether the other node has higher active degree. The processing proceeds to step (4) if yes to either of above check and step (5) otherwise (line 10-21).

(4) The node forwards the packet to the other node. When the other node receives the packet, if the storage is full, the oldest packet is dropped until the available memory can hold the new packet. Then, if the new packet is not dropped, the node inserts its ID into the packet and stores the packet. (line 25-33).

(5) The process of the current packet stops. If there are unprocessed packets, the checking process repeats from step (2) for the next packet (line 7).

Algorithm 1 Pseudo-code of the SMART routing algorithm executed by a upon meeting node b.

```
 1: procedure EXCHANGETOPFRIENDSWITH(b)
 2:     n.sendTopFriendsTo(b)
 3:     n.receiveTopFriendsFrom(b)
 4:     n.updateSocialMap()
 5: end procedure
 6: procedure SELECTFORWARDER(b)
 7:     for each packet p in node a do
 8:         bForward ← false
 9:         if p.hasBeenOn(b) = false then
10:             if a.connect(p.des) || b.connect(p.des) then
11:                 if b.getW(p.des) < a.getW(p.des) then
12:                     bForward ← true
13:                 end if
14:             else
15:                 if b.getDegree( ) > a.getDegree( ) then
16:                     bForward ← true
17:                 end if
18:             end if
19:             if bForward = true then
20:                 a.forwardPacketTo(p, b)
21:             end if
22:         end if
23:     end for
24: end procedure
25: procedure RECEIVEPACKETSFROM(p, b)
26:     while Memory.Full() = true do
27:         dropOldestPacket()
28:     end while
29:     if p.NotDropped() = true then
30:         p.InsertID(a)
31:         StorePacket(p)
32:     end if
33: end procedure
```

In summary, two encountering nodes in SMART only exchange a small amount of infor-

mation for social map construction and forwarder selection. Also, the delivery ability in SMART naturally considers the multi-hop relay through the top L friends. This global view based forwarder selection can enable more efficient routing. SMART can also support different routing metrics such as minimal average/maximal delay and minimal missed deadlines by setting different packet forwarding priorities when two nodes meet [9]. SMART uses first-come-first-out forwarding sequence in this dissertation.

5.3.3 Effect of Top L Friends

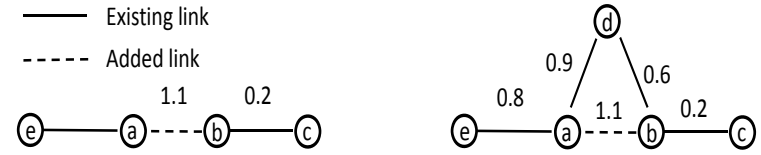
Recall that SMART only allows each node to keep a node's top L friends in the social map. We now analyze how L affects the routing efficiency and further check the correctness of our design (i.e., only store top L friends) based on the routing procedure.

As aforementioned, the key step in the routing procedure is to find the minimal weight path from the candidate forwarders to the destination in the social map. The precision of discovered minimal weight path decides the effectiveness of the forwarder selection and consequently, the routing efficiency. We discuss the scenario when we gradually increase L from 1. When L increases, each node can initiate more links in the social map. The added links can help find a better minimal weight path (i.e., with a smaller weight) by two ways:

- It connects two previously disconnected nodes and enables a new minimal weight path, as shown in Figure 5.7(a).
- It improves an existing minimal weight path by lowering its weight, as shown in Figure 5.7(b).

Clearly, when L is small, the two cases happen easily if L increases, thereby improving the accuracy of discovered minimal weight path. However, when L is large, we argue that both the two cases are not easy to happen.

Firstly, when L is large enough, the graph is almost connected. The first case then can hardly happen. Secondly, for the second case, by examining Figure 5.7(b), we find that it happens when the weight of the added link (1.1) is smaller than the sum of the weights of the two previous links connecting the two nodes ($0.9 + 0.6$). Then, since the links are added incrementally, i.e., the newly added link for a node is its (L+1)-th friend when L increases to $L + 1$, when L is large, the weights of the new links are large, thereby can hardly satisfy the requirement for the second case.



(a) Connecting disconnected nodes.

(b) Improve existing path (e to c).

Figure 5.7: Improvement on social map when L increase.

In conclusion, when L increases from a small value, the two cases happen easily and thus improve the routing efficiency. However, when L is larger enough, increasing it can hardly trigger the two cases to improve the routing efficiency. Such a result demonstrates the correctness of only storing top L friends (with a proper L). Such a finding matches our discussion in Section 5.2.1 that people forward messages mainly through major social relationships.

5.3.4 Advanced Extensions of Packet Routing

We present two extensions of SMART that can improve the routing efficiency.

5.3.4.1 Alleviating the Load on Overloaded Nodes

In the routing protocol, active nodes that are a top L friend of more nodes have more chances to be selected as the packet forwarder, and hence can easily become overloaded. To avoid overloading such nodes, we introduce an overload bit in the beacon messages. When a node is about to be overloaded, it sets the overload bit in its beacon messages. Then, its neighbors will not select it as a packet forwarder candidate. The overload bit is reset when the load on the node recovers to a normal level.

5.3.4.2 Reducing Computation Cost

Recall that when a needs to decide whether the newly met node b is a better forwarder for one of its packet, the Dijkstra algorithm regards the destination as the root node and iteratively selects the node that has the minimal path weight to the destination. The selection process ends when either a or b is picked up, and the first picked node in a and b is a better carrier for the packet. The complexity of this process is $O(n^2)$ [2], where n is the number of nodes in the social map. This process is executed for the destination of each packet in node a. Therefore, the computation load is

very high without further optimization.

In order to reduce the computation load, we propose to proactively cache better forwarders for each destination node. Specifically, for each destination, node a runs the Dijkstra algorithm until a is picked or all nodes are picked. Then, the nodes that are selected prior to a and their path weights to the destination are stored in a Better Forwarder Table (BFT). Thus, the BFT records the nodes that have higher delivery ability to the destination node than node a. Table 5.3 shows an example of a BFT in node a. Later on, when node a encounters these recorded nodes, it forwards the packets for corresponding destination to them directly without running the Dijkstra algorithm. If multiple nodes in BFT for the same destination are met at the same time, the one with the minimal path weight is selected as the forwarder.

Table 5.3: Better forwarder table (BFT) in node a

Destination	Better forwarder (path weight)
b	$k(1.8), e(1.7), h(0.4), o(0.9)$
c	$b(2.2), c(1.2), j(1.4), f(0.8)$
f	$k(0.8), m(0.9), d(1.3)$
$\cdots$	$\cdots$

The BFT should be updated timely to reflect the changes on the social map due to meeting frequency change in MONs. It is updated when either the destination of a packet is absent from the table or the number of changed social links constitutes more than $TH_c\%$ of all social links in the social map. Note that each node records the number of added or removed social links in its social map to track the second case. When the former happens, the Dijkstra algorithm is launched to make the forwarding decision for the destination and meanwhile update the corresponded row in the BFT. When the latter happens, all entries in the BFT become invalid and are removed, after which the BFT is updated following the first scenario.

We then discuss the maintenance and storage cost of the BFT. Firstly, the BFT table updates frequently at the beginning when the social map has less information. However, as mentioned in Section 5.2.1, a node's most frequently met nodes are stable. Thus, after the initial stage, the social map on a node would become stable. Then, we can set a relatively large TH_c, which would lead to a low update overhead for BFT. The BFT in a node only stores the IDs of better forwarders and their associated path weights for each destination. Since a pair of ID and weight only occupies several bytes, the size of a BFT would not be a burden to modern mobile devices which usually have GB level memory (i.e., 8 GB).

Algorithm 2 shows the process for node a to decide the forwarder for its packets when it meets node b. For each of its packets, node a first checks if both a and b are disconnected to the packet's destination in the social map. If so, the forwarder should be the one with the higher active degree. Otherwise, node a checks whether the entry for the destination node exists in its BFT and node b exists in the entry. If so, it forwards the packet to node b directly. If not, node a uses the Dijkstra algorithm to decide the forwarder as previously described in Section 6.2.4, and meanwhile updates its BFT.

Algorithm 2 Pseudo-code of the SMART BFT-based routing algorithm executed by a upon meeting node b.

```
1: procedure SELECTFORWARDER(b)
2:     for each packet p in node a do
3:         d ← destination of packet p
4:         if both a and b are disconnected from d then
5:             Forwarder(p)←the node in {a, b} with higher active degree
6:         else
7:             if there is an entry for d in BFT then
8:                 if b is in the entry then
9:                     Forwarder(p)←b
10:                end if
11:            else
12:                LaunchDijkstraAlgFor(p)
13:            end if
14:        end if
15:    end for
16: end procedure
```

5.4 Discussion on Scalability and Security

We also briefly discuss the scalability and security issues of SMART.

5.4.1 Scalability of SMART

Although each node in SMART needs to store many friend maps, SMART is scalable on storage in a large-scale network with 10,000 nodes due to two reasons. First, one friend map only contains L IDs and L friendship ranks, which occupy about $8L$ bytes. Then, 10,000 friend maps require about $80L$ KB memory, which is not a big burden for most mobile devices nowadays. Second, a node only stores the friend maps of nodes it has met, which are very limited in a large network since it mostly only meets nodes in the local community. Therefore, even when the network size is very large, the storage consumption in SMART is limited. As mentioned in the introduction section, SMART is designed mainly for applications in a certain local community, which means the network

size usually is not very big. In our experiment, both traces contain less than 100 nodes. Then, if $L = 8$, the social map on a node needs at most 6.4 KB. Thus, the memory can be satisfied easily in potential application scenarios.

With BFT, the Dijkstra algorithm runs only when necessary. The complexity to determine packet forwarder by checking the better forwarder table is $O(n)$. Recall that without BFT, the complexity to determine packet forwarder using the Dijkstra algorithm is $O(n^2)$. Therefore, the computation complexity is greatly reduced with BFTs. Our experimental results in Section 5.5.4 show that the BFT greatly reduces the computation cost without greatly compromising the routing efficiency.

SMART is also scalable regarding routing path lengths. Large social networks have a 6-hop property, in which two unknown persons can be connected by 6 hops of transits on average [83]. This implies that through top L friends, SMART may not need a large number of hops to deliver a packet to its destination even when the network is very large. Note that SMART is designed for social networks in local communities where the connections between people usually are tight. Thus, in SMART, a node needs fewer hops to reach an unknown person. Our experimental results also show that the average number of forwarding hops for a successfully delivered packet is about 5 in tests with both real traces.

5.4.2 Security in SMART

In MONs, nodes may belong to different entities or organizations. Therefore, some nodes may not follow the SMART routing algorithm or even behave maliciously for individual interests. We then briefly discuss the blackhole attack in SMART.

In blackhole attack, malicious node a claims that node b is its friend and falsely report a very high friendship rank with b in order to attract packets destined to b and drop these packets. We propose to use friendship verification to detect and prevent it. Specifically, when node a wants to take node b as its top L friend, node a must send the friendship rank to node b asking for an approval. Node b then verifies the correctness of the friendship rank and signs the friendship rank with its private key if it is correct. The friendship rank can be updated and signed timely since they meet frequently (i.e., b is a top L friend of a). Later, when a claims b as its top L friends, other nodes can verify the friendship rank with the public key of node b. As a result, nodes that have forged the friendship rank can be identified, thereby preventing the blackhole attack.

An advantage of the above method is that a node would not collude with another node to fake the friendship ranks since this would reduce its opportunity to receive packets destined for it. However, a node may attract and drop packets by its real friendship ranks. The detection of such behaviors is non-trivial in MONs and is out of the scope of this dissertation.

5.5 Performance Evaluation

We conducted event-driven experiments using real traces from the MIT Reality project [34] and the Haggle project [18]. We first focus on the test with fixed L to show how different L affects the routing performance. Then, we examine the performance of SMART under different Ls. We also examined the performances of the method to decide L distributively and the better forwarder table (BFT). We compared SMART with following representative MON routing algorithms.

(1) *PROPHET*: PROPHET is a probabilistic routing algorithm. It calculates delivery ability based on past encountering records and forwards packets to nodes with higher delivery ability to their destinations.

(2) *SimBet*: SimBet is a social network based algorithm. It calculates the suitability of a node for carrying a packet by the node's centrality value and its similarity with the destination node (the number of shared encountered nodes). Packets are always forwarded to nodes with better suitability.

(3) *StaticWait*: In StaticWait, a source node carries its packet until meeting the packet's destination. We use this algorithm as a baseline method to show the routing efficiency when no active forwarding strategy is adopted.

In the experiment, the first 1/3 of both traces were used as the initialization period to collect enough encountering records. After this, packets were generated at the rate of R_n per 300s and per 40s in the MIT Reality trace and the Haggle trace, respectively. In the test, at most $10 * T_l$ packets can be exchanged when two nodes meet, where T_l is the length of the encountering session in seconds. The size of a packet was set to 1 KB. The source and destination of a packet were randomly selected from all nodes in the system. In SMART, L was set to 4 by default. In PROPHET and SimBet, the parameters used to calculate the delivery ability and utility were configured to the same values as in their papers. We used the packet replacement strategy in SMART, PROPHET, and SimBet.

We tested the performance of SMART with different packet rates, different memory sizes

on each node, and different values of L. In the test with different packet rates, the total number of packets was varied from 5,000 to 25,000 (i.e., R_n was varied from 1 to 5). The memory size on each node was set to 100KB. In the test with different memory sizes, the memory on each node was varied from 60KB to 140 KB with an increase of 20 KB in each step, and the packet rate (R_n) was set to a medium value of 3. In the test with different values of L, we varied the value of L from 2 to 8, and set the packet rate to 3 and memory size on a node to 100 KB.

We measured the following metrics during the test.

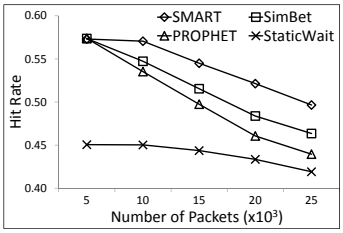
- *Hit rate:* the percentage of requests that are successfully delivered to their destinations in the experiment.
- *Normalized average delay:* the average delay of all packets. We regard the delay of an unsuccessful packet as the trace length, which is 340k and 2560k seconds for the Haggle trace and the MIT Reality trace, respectively.
- *Normalized forwarding hops:* the total number of packet forwarding hops divided by the number of successfully delivered packets.
- *Routing cost:* the number of information units exchanged between encountered nodes in the experiment.

5.5.1 Performance with Different Packet Generating Rates

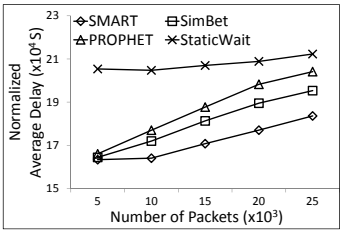
5.5.1.1 Hit Rate

Figure 5.8(a) and Figure 5.9(a) demonstrate the hit rates of the four methods with the Haggle trace and the MIT Reality trace, respectively. From the two figures, we see that the hit rates of the four methods follow SMART>SimBet>PROPHET>StaticWait.

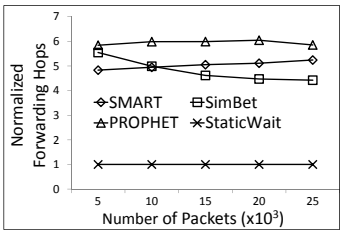
StaticWait shows the lowest hit rate because packets only statically wait in their generators to reach destinations. SMART deduces nodes' delivery abilities to destinations based on the relatively stable social map. It can choose an optimal forwarder in a long path to the destination with a broad view, thereby generating the highest hit rate. PROPHET and SimBet can only evaluate the delivery ability within one or two hops, which cannot consider long routing paths, leading to a lower hit rate than SMART.



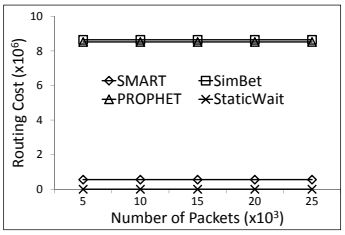
(a) Hit rate.



(b) Normalized average delay.

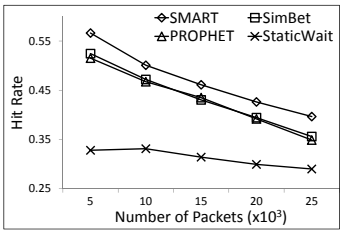


(c) Normalized forwarding hops.

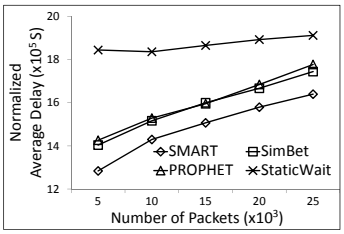


(d) Routing cost.

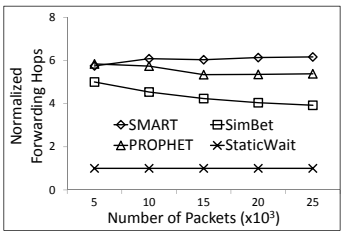
Figure 5.8: Performance of each method with the Haggle trace under different packet rates.



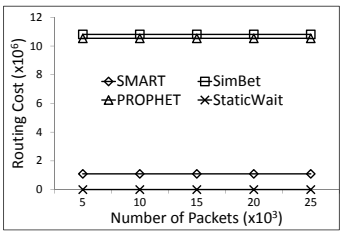
(a) Hit rate.



(b) Normalized average delay.



(c) Normalized forwarding hops.



(d) Routing cost.

Figure 5.9: Performance of each method with the MIT Reality trace under different packet rates.

We observe that as the total number of packets increases, the hit rates of the four methods decrease. This is because the forwarding opportunities and memory available for packet forwarding are limited. Then, when more packets are generated, more packets are dropped by nodes, leading to a decreased hit rate. It is interesting to see that the hit rates of the three methods with active forwarding decrease more quickly than that of StaticWait. This is because we configure fixed memory in the test. Then, when the number of packets increases, more packets are converged to certain important nodes, which have fixed storage, resulting in more dropped packets. We can adopt load balance techniques in Section 5.3.4 to alleviate this problem. In StaticWait, packets are evenly distributed among nodes, thereby better utilizing the memory on all nodes and leading to fewer dropped packets.

With above results, we conclude that SMART can achieve efficient routing in the MON environment with the proposed social map. These results also justify the correctness of the design of the link weight calculation method that considers both meeting frequency and social closeness.

5.5.1.2 Normalized Average Delay

Figure 5.8(b) and Figure 5.9(b) show the normalized average delays of the four methods in the tests with the Haggle trace and the MIT Reality trace, respectively. From the two figures, we find that the normalized average delays of the four methods follow SMART<SimBet<PROPHET< StaticWait. SMART has the lowest normalized average delay because it considers multi-hop forwarding opportunities when making forwarding decisions with a broader view from the social map, which enables a packet to travel through a fast route to its destination. Both PROPHET and SimBet fail to consider long routing paths that may generate shorter delay. Therefore, they produce higher average delay than SMART. StaticWait has the highest average delay since packets only wait in their initiators for destinations without being forwarded. Such results further demonstrate the high efficiency of SMART in terms of routing delay.

We also find that the normalized average delays of the four methods increase as the packet generating rate increases. This is because we regard the delay of dropped packets as the length of the test trace, which is much longer than the delay of a successful packet. Then, when the packet generating rate increases, there are more dropped packets (the hit rate decreases), as shown in Figure 5.8(a) and 5.9(a), leading to increased normalized average delay.

5.5.1.3 Normalized Forwarding Hops

Figure 5.8(c) and Figure 5.9(c) show the normalized forwarding hops of the four methods with the Haggle trace and the MIT Reality trace, respectively. We observe that StaticWait has very low forwarding hops and other three methods have high forwarding hops. In StaticWait, each packet waits in its initiator for the destination. Therefore, each successful packet is forwarded only once. The active forwarding in SMART, SimBet and PROPHET leads to much more forwarding for each successful packet. We see that the normalized forwarding of SMART is on the same level with SimBet and PROPHET, but SMART has higher hit rate and lower average delay. This further demonstrates that the active forwarding in SMART is more effective.

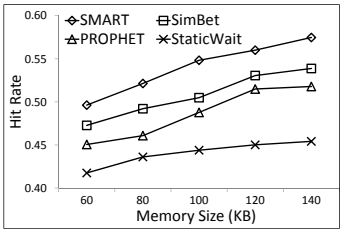
5.5.1.4 Routing Cost

Figure 5.8(d) and Figure 5.9(d) plot the routing costs of the four methods with the Haggle trace and the MIT Reality trace, respectively. We see that StaticWait has no routing cost since no information exchange is needed. SMART incurs a significantly lower routing cost than PROPHET and SimBet. This is because two encountered nodes only need to exchange their friend maps with L (usually a small value) entries in SMART, while nodes need to exchange the information regarding the destination nodes of all packets in PROPHET and SimBet. SimBet has a slightly higher routing cost than PROPHET since in addition to the similarity information, a node has to send its centrality information to the newly met node. In a nutshell, StaticWait incurs the least total costs but has a low efficiency, SMART consumes low total transmission and storage costs, and PROPHET and SimBet generate very high total transmission and storage cost. This result confirms SMART's low-cost on information exchange.

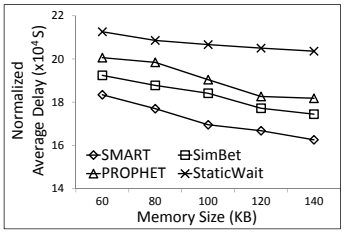
5.5.2 Performance with Different Memory Sizes on Each Node

5.5.2.1 Hit Rate

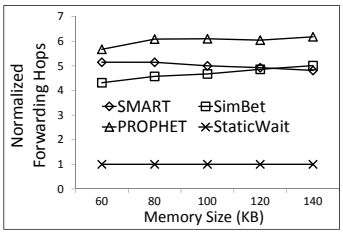
Figure 5.10(a) and Figure 5.11(a) demonstrate the hit rates of the four methods with the Haggle trace and the MIT Reality trace when the memory size varies, respectively. We see that the hit rates of the four methods follow the same as in Figure 5.8(a) and Figure 5.9(a) due to the same reasons. We also find that when the memory size on a node increases, the hit rates of all methods also increase. This is because with larger memory size, each node can carry and forward



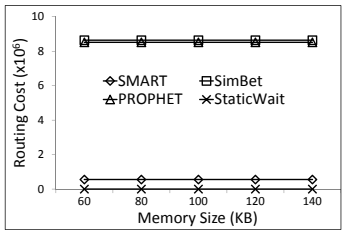
(a) Hit rate.



(b) Normalized average delay.

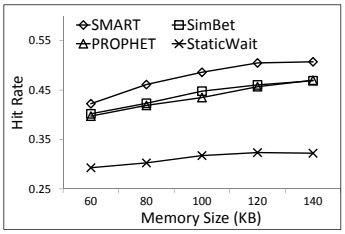


(c) Normalized forwarding hops.

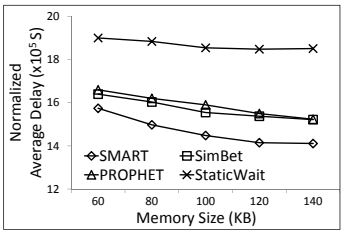


(d) Routing cost.

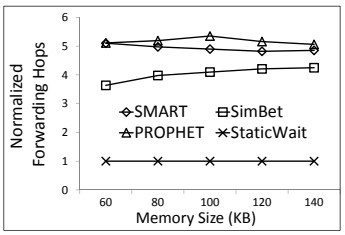
Figure 5.10: Performance of each method with the Haggle trace under different memory sizes.



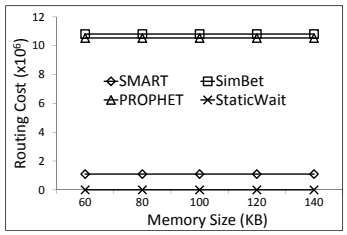
(a) Hit rate.



(b) Normalized average delay.



(c) Normalized forwarding hops.



(d) Routing cost.

Figure 5.11: Performance of each method with the MIT Reality trace under different memory sizes.

more packets to their destinations, leading to a higher hit rate.

5.5.2.2 Normalized Average Delay

Figure 5.10(b) and Figure 5.11(b) show the average delay of the four methods with the Haggle trace and the MIT Reality trace when the memory size on each node varies, respectively. We find that the relationship of the four methods on average delay is the same as in Figure 5.8(b) and Figure 5.9(b) for the same reasons. It is interesting to see that when the memory size on a node increases, the normalized average delays decreases. This is because when the memory size increases, there are fewer dropped packets, whose delay is very large (i.e., trace length), leading to decreased normalized average delay.

5.5.2.3 Normalized Forwarding Hops

Figure 5.10(c) and Figure 5.11(c) show the number of forwarding hops of the four methods with the Haggle trace and the MIT Reality trace when the memory size on each node varies, respectively. We find that the normalized forwarding hops follow the same trend as in Figure 5.8(c) and Figure 5.9(c) due to the same reasons.

5.5.2.4 Routing Cost

Figure 5.10(d) and Figure 5.11(d) demonstrate the routing costs of the four methods with the Haggle trace and the MIT Reality trace when the memory size on each node varies, respectively. We find that the routing costs are the same as the Figure 5.8(d) and Figure 5.9(d) for the same reasons. This is because the routing cost is irrelevant to the number of packets on each node but is only related to the L for SMART and the number of nodes in the system for other three methods. Combining all the above results, we conclude that SMART has superior performance than other methods in MON routing with different memory sizes on each node.

5.5.3 Effect of the Value of L

In this section, we varied the value of L used in SMART from 2 to 8 to evaluate its effect on the routing performance and verify that large L values (i.e., greater than a threshold) would not significantly enhance the routing performance. The total number of packets was set to a medium value of 15,000. The results are shown in Table 5.4 and Table 5.5.

Table 5.4: Routing performance with the Haggle trace

L	Hit Rate	N. Ave. Delay (s)	N. Fwd. Hops	Routing Cost
2	0.541425	172,614.8	4.992	277,704
3	0.551023	169,006.2	4.932	416,556
4	0.551156	168,715.7	5.050	555,408
5	0.560021	165,903.7	4.945	833,112
6	0.56822	163,745.5	4.860	833,112
7	0.572219	162,426.5	4.825	971,964
8	0.576218	161,251.9	4.775	1,110,816

Table 5.5: Routing performance with the MIT Reality trace

L	Hit Rate	N. Ave. Delay (s)	N. Fwd. Hops	Routing Cost
2	0.431467	1,575,819.5	4.028	546,892
3	0.459933	1,509,557.2	4.776	820,338
4	0.481467	1,459,355.8	4.858	1,093,784
5	0.502867	1,413,748.9	4.884	1,367,230
6	0.502533	1,410,643.2	5.027	1,640,676
7	0.515667	1,384,143.6	4.994	1,914,122
8	0.515933	1,385,610.1	4.955	2,187,568

5.5.3.1 Hit Rate

We see that the hit rate of SMART increases steadily when L increases from 2 to 8. This is because the calculation of link weight depends on the number of shared top L friends. Therefore, when L increases, the calculated weight can reflect the delivery ability of the two connected nodes more precisely. Consequently, a node can more correctly decide if a newly met node is a better carrier for its packets, leading to improved routing efficiency. This result confirms the feasibility of constructing a social map by exchanging only the top L friends in MONs and implies that L can be adjusted to achieve a tradeoff between cost and routing efficiency.

We also see that the increase of hit rate with the MIT Reality trace is much larger than that with the Haggle trace when L increases. This is because the two traces were obtained in different environments. The Haggle trace was conducted in a crowded conference scenario, in which each node can meet many nodes frequently; so a small L can still approximately represent the social structure and will not decrease the hit rate significantly. However, the MIT Reality trace was obtained in a sparse campus environment. In this case, a small L would lose some important friends, thereby providing fewer forwarding opportunities and leading to a low hit rate.

5.5.3.2 Normalized Average Delay

We find that the normalized average delay of SMART decreases when L increases at the beginning and remains at the same level when L is larger than a medium value (i.e., 5). When the

L increases from a small value, the social map constructed on each node becomes more complete, resulting in better forwarder selection and decreased average delay. The increased hit rate also leads to fewer dropped packets, which have high delay (i.e., trace length). Consequently, the normalized average delay decreases. After L reaches the medium value, which enables social maps to show almost all most frequently met nodes, the enhancement in the routing efficiency is not significant if L further increases, as shown in the first column of the two tables. Therefore, the normalized average delays remain on the same level.

5.5.3.3 Normalized Forwarding Hops

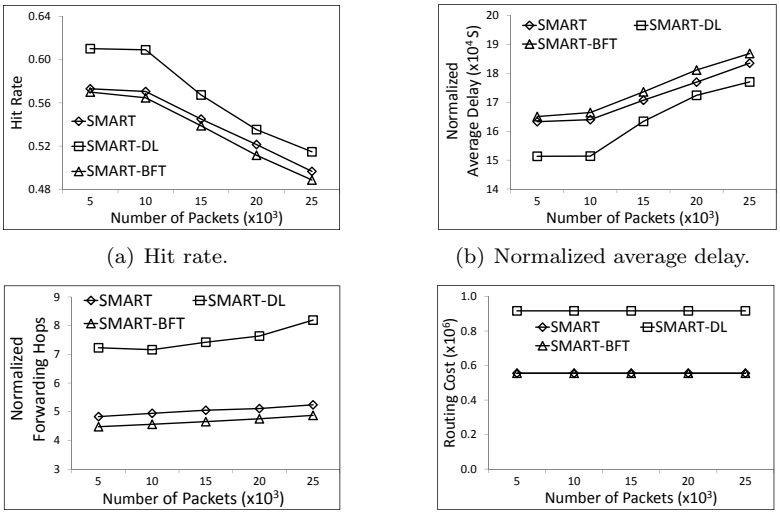
We see that when L increases from 2 to 8, the normalized forwarding hops of SMART remains stable when the Haggle trace is used and increases in the test with the MIT Reality trace. This is caused by the different environments of the two traces. In the Haggle project, nodes are more crowded, so it is easy to find a next-hop node even when L is small. Therefore, the normalized forwarding hops remain stable with different L. But in the MIT Reality trace, nodes are sparser, so a small L cannot reflect the social structure well, leading to fewer forwarding opportunities and a low normalized forwarding hops.

5.5.3.4 Routing Cost

We find that the routing cost increases in proportion to the value of L when it increases from 2 to 8 with both traces. This is because the routing cost is actually the number of encounters multiplied by $2L$. We see that the routing cost is still quite small when L is 8 compared to that of SimBet and PROPHET shown in Figure 5.8(d) and Figure 5.9(d). This result shows the efficiency of SMART in reducing information transmission, and also implies that it is important to find an optimal L value that generates low routing cost while achieving high routing efficiency.

5.5.3.5 Summary

Above experimental results indicate that SMART still works efficiently when L is set to a small value. For example, even when $L = 2$, SMART still generates close performance on hit rate, average delay, and cost with other compared methods. Also, the performance of SMART is improved when L increases and remains stable when L is larger than 5. Such a result matches our analysis in Section 5.3.3 and justifies the idea that top L friends can provide enough critical information to



(a) Hit rate.

(b) Normalized average delay.

(c) Normalized forwarding hops.

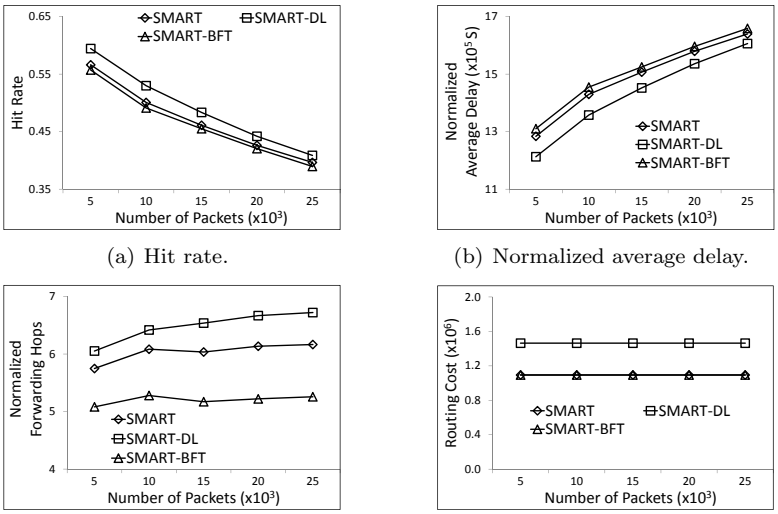
(d) Routing cost.

Figure 5.12: Performance of each extension with the Haggle trace under different packet rates.

guide packet forwarding. SMART achieves an optimal balance between efficiency and cost when L is set to a medium value such as 4 or 5 for the two traces. This value may change in different scenarios. However, since the increase in the routing cost is linear when L increases, we conclude that SMART is energy efficient and suitable for MONs.

5.5.4 Effect of Dynamic L and BFT-based Routing

In Section 5.2.1.4, we proposed two methods to determine L: the centralized method and the distributed method. In the former, L is pre-determined based on network statistics and all nodes have the same L. In the latter, L is dynamically decided on each node. In Section 5.5.3, we already presented the performance of SMART with different L values in the centralized method. In this section, we present the performance of SMART with dynamically determined L values in the distributed method, denoted by SMART-DL. In SMART-DL, L is determined so that the L-th most frequently met node has half of the meeting frequency of the most frequently met node. We also present the performance of the BFT-based routing introduced in Section 5.3.4.2, denoted by SMART-BFT. In the test, we set the memory on each node to a medium value of 100KB, L to a medium value of 4 in SMART and SMART-BFT, and the threshold of the percentage of changed links in a social map for BFT update in SMART-BFT to 20%.



(a) Hit rate.

(b) Normalized average delay.

(c) Normalized forwarding hops.

(d) Routing cost.

Figure 5.13: Performance of each extension with the MIT Reality trace under different packet rates.

5.5.4.1 Hit Rate

Figures 5.12(a) and 5.13(a) show the hit rates of SMART, SMART-DL and SMART-BFT with the Haggle trace and the MIT Reality trace, respectively. We see that in both traces, SMART-DL achieves higher hit rate than SMART. This is because in SMART-DL, dynamically determined Ls can reflect the major social relationships in the social map, leading to a higher hit rate. We found that the average value of L equals around 5-6 in SMART-DL, which is larger than $L = 4$ in SMART. We also see that SMART-BFT has a slightly lower hit rate than SMART. The BFT in SMART-BFT is updated only when the percentage of the changed links in a social map is larger than a threshold (i.e., 20%), leading to insufficiently accurate routing guidance for some packets and hence slightly degraded hit rate. However, the hit rate decrease is very minor while SMART-BFT significantly reduces the computation cost in forwarder determination in the routing process, as shown below.

5.5.4.2 Normalized Average Delay

Figures 5.12(b) and 5.13(b) show the normalized average delays of SMART, SMART-DL and SMART-BFT with the Haggle trace and the MIT Reality trace, respectively. We find that SMART has larger average delay than SMART-DL in the tests with both the traces. This is because the social maps in SMART-DL can reflect more social relationships. Therefore, it can guide the packet forwarding more accurately, leading to lower average delay. We also see that SMART-BFT has

slightly larger average delay than SMART. For SMART-BFT, it cannot reflect the changes on social maps timely, which lowers the accuracy of forwarder selection and increases the average delay.

5.5.4.3 Normalized Forwarding Hops

Figures 5.12(c) and 5.13(c) show normalized forwarding hops of SMART, SMART-DL and SMART-BFT with the Haggle trace and the MIT Reality trace, respectively. We see that in both traces, the normalized forwarding hops follows SMART-DL>SMART>SMART-BFT. With the dynamic L, the social maps in SMART-DL can show more close social relationships, leading to more packet forwarding opportunities and consequently more packet forwarding. On the contrary, the untimely updated better forwarder table in SMART-BFT reduces the number of packet forwarding opportunities, leading to less packet forwarding than SMART. These results match those in Figure 5.12(a) and 5.13(a) since higher hit rate comes at more forwarding hops.

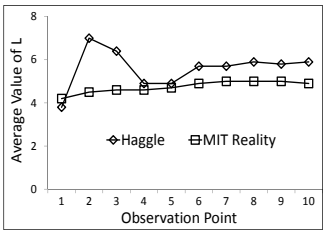
5.5.4.4 Routing Cost

Figures 5.12(d) and 5.13(d) show the routing costs of SMART, SMART-DL and SMART-BFT with the Haggle trace and the MIT Reality trace, respectively. We find that SMART-BFT and SMART generate the same routing cost and SMART-DL produces higher routing cost. Both SMART-BFT and SMART have fixed $L = 4$, leading to the same routing cost. Nodes in SMART-DL dynamically adjust L. We found that the average values of L in SMART-DL is around 5-6, leading to more information exchange among top L friends.

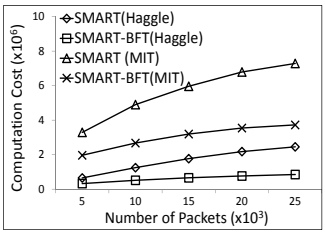
Figure 5.14(a) shows the average values of L at 10 observation points, which are evenly distributed in each trace. We find that after half of each trace, the average values of L are about 6 and 5 in the Haggle trace and the MIT Reality trace, respectively. This result shows the average number of stable friends of each node in the two traces. Comparing the results in Figure 5.12(a) and Figure 5.13(a) and those in Table 5.4 and Table 5.5, we find that SMART-DL achieves higher hit rate than SMART with fixed $L = 6$ and $L = 5$. This result demonstrates that the proposed method to determine L dynamically can effectively find the important social relationships in the network.

5.5.4.5 Computation Cost Reduction in SMART-BFT

We further evaluate the effect of SMART-BFT on reducing the computation cost with different packet generating rates. Since the Dijkstra algorithm used in the routing has the most



(a) Average value of L.



(b) Computation cost.

Figure 5.14: Experiment results on average value of L and computation cost.

computation cost, we measure the computation cost as the number of Dijkstra algorithm launched in the experiment. The test results are shown in Figure 5.14(b), in which the name after a method denote the trace used in the experiment. We find that the BFT can save up to 65% of computation cost in the tests with both traces. Combining the results in Figure 5.12 and Figure 5.13, we conclude that the BFT-based routing can significantly reduce the computation cost of SMART without greatly compromising the routing efficiency.

5.6 Summary

In this chapter, we propose SMART, which is a lightweight routing algorithm in mobile opportunistic networks that utilizes distributed social maps on mobile nodes. By exploiting the social network property that a person's most frequently encountered friends often remain stable, SMART enables each node to build a social map to record its knowledge of surrounding social structure. Specifically, nodes exchange the top L most frequently encountered nodes when they meet for social map construction. In the social map, the delivery ability between two nodes is evaluated by considering both meeting frequency and social closeness. Then, packets are forwarded to nodes with higher delivery ability to their destinations. SMART is more efficient than previous routing algorithms because the social map offers a broader view for forwarder selection. Moreover, two encountering nodes only need to exchange the information of their top L friends, which is relatively stable, leading to a low information exchange and update overhead. Extensive real-trace driven experiments demonstrate the effectiveness of SMART.

Chapter 6

Exploiting Node Mobility for High-Throughput Packet Routing among Landmark in MONs

While the previous chapter proposes a novel algorithm for efficient packet routing among nodes in MONs, we further investigate how to realize high-throughput packet routing among landmarks in MONs. Considering people usually present certain mobility pattern when transiting among different landmarks, we propose an inter-landmark packet routing algorithm, called DTN-FLOW, to fully utilizes all node movements in MONs. DTN-FLOW selects popular places that nodes visit frequently as landmarks and divides the entire MON area into sub-areas represented by landmarks. DTN-FLOW then measures the amount of nodes moving from one landmark, say L_i, to another landmark, say L_j, to represent the packet forwarding capacity from L_i to L_j. This capacity indicates the packet transfer capacity between landmarks, hence is similar to the concept of "bandwidth" for physical wired or wireless links. With the measured capacity, each landmark uses the distance-vector method [47] to build its routing table that indicates the next hop landmark on the fastest path to reach each destination landmark. DTN-FLOW predicts node transits based on their previous landmark visiting records using the order-k Markov predictor. In packet routing, each landmark determines the next hop landmark for each packet based on its routing table, and forwards the packet to the node with the highest probability of transiting to the selected landmark. Thus, DTN-FLOW

fully utilizes node transits among landmarks to forward packets along landmark paths that have the shortest latency to reach their destinations.

In the following, we first present network model of DTN-FLOW and the analysis of two real MON traces to confirm the shortcoming of the current routing algorithms and to support the design of DTN-FLOW in Section 6.1. We then introduce the detailed design of DTN-FLOW in Section 6.2. The performance evaluation is given out in Section 6.3. Finally, Section 6.4 summarizes this chapter.

6.1 Network Model and Trace Analysis

We first introduce the network model of the proposed system. We then analyze two real traces to support the design of DTN-FLOW.

6.1.1 Network Model

6.1.1.1 Network Description

We assume a MON with mobile nodes denoted by N_i. Each node has limited storage space and communication range. We select landmarks, denoted by L_i ($i = 1, 2, 3, \cdots, M$), from places that nodes visit frequently. Then, the entire MON area is split into sub-areas based on landmarks, each of which is represented by a landmark. We configure a central station at each landmark, which has higher processing and storage capacity than mobile nodes and can cover its whole sub-area. As in other social network based MON routing algorithms [10, 29, 30, 53, 68, 92], DTN-FLOW also assumes the existence of social network structure in MONs. Such social structures determine node movement, leading to re-appearing visiting patterns to these landmarks.

A transit means a node moves from one landmark to another landmark. We denote the *transit link* from landmark L_i to landmark L_j as T_{ij}. For a transit link, say T_{ij}, we define the *bandwidth* as the average number of nodes transiting from L_i to L_j in a unit time (T), denoted by B_{ij}. For simplicity, we assume that each packet has a fixed size. Our work can be easily extended to packets with various lengths by dividing a large packet into a number of the same-size segments.

6.1.1.2 Differences with Infrastructure Networks

Though DTN-FLOW presents similar overlay as the infrastructure network (i.e., sub-areas covered by landmarks), they have significant differences. Firstly, DTN-FLOW does not require landmarks to be inter-connected with fixed links. Rather, landmarks rely on the mobile nodes moving between them to relay packets. Secondly, as shown later, a landmark only functions as a special relay node in the packet routing. Therefore, landmarks do not bring about server-client structure but keep the ad hoc nature of MONs. Unlike base stations, landmarks are just static nodes in MONs with higher processing capacity.

6.1.1.3 Purpose of Landmarks

In DTN-FLOW, landmarks function as "routers" in the network. Each landmark decides the neighbor landmark to forward its received packets. Neighbor landmarks are connected by "links" that take mobile nodes as the transfer media for packets. Without landmarks, packets are relayed purely through mobile nodes when they meet with each other, which may suffer from the uncertainty of node mobility and the limited number of nodes that can deliver them quickly. Therefore, landmarks make the MON routing more structured (i.e., along landmarks) under the network dynamism in MONs. In summary, the use of landmarks in DTN-FLOW can better utilize node mobility in MONs for efficient packet routing with a low extra cost.

6.1.2 Trace Analysis

In order to better understand how nodes transit among different landmarks in MONs, we analyzed two real MON traces collected from two different scenarios: students on campus and buses in the downtown area of a college town.

6.1.2.1 Empirical Datasets

Dartmouth Campus Trace (DART) [48]. DART recorded the WLAN Access Point (AP) association with digital devices carried by students in the Dartmouth campus between Nov. 2, 2003 and Feb. 28, 2004. We preprocessed the trace to fit our investigation. We regarded each building as a landmark and merged neighboring records referring to the same node (mobile device) and the same landmark. We also removed short connections ($< 200s$) and nodes with few records

(< 500). Finally, we obtained 320 nodes and 159 landmarks.

DieselNet AP Trace (DNET) [8]. DNET collected the AP association records from 34 buses in UMass Transit from Oct. 22, 2007 to Nov. 16, 2007 in Amherst, MA. Each bus carried a Diesel Brick that constantly scanned the surrounding area for open AP connections, and a GPS to record its GPS coordinators. Since there are many APs in the outdoor testing environment, some of which are not from the experiment, we removed APs that did not appear frequently (< 50) from the trace. We mapped APs that are within certain distance ($< 1.5km$) into one landmark. Similar to the processing of the DART trace, neighboring records refereeing to the same node (bus) and the same landmark were merged. Finally, we obtained 34 nodes and 18 landmarks.

The key characteristics of the two traces are summarized in Table 6.1. We then measured the landmark visiting distribution and the transits of mobile nodes among landmarks.

Table 6.1: Characteristics of mobility traces.

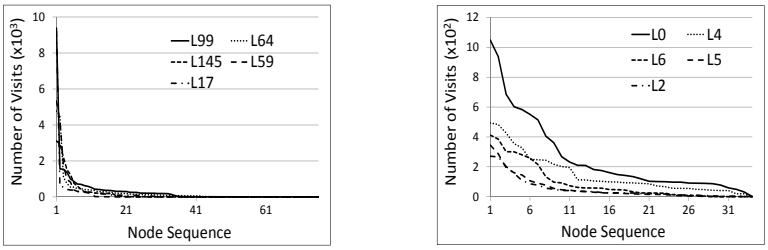
	DART Campus (DART)	DieselNet AP (DNET)
# Nodes	320	34
# Landmarks	159	18
Duration	119 days	20 days
# Transits	477803	25193
# Transits per day	2401	1257

6.1.2.2 Landmark Visiting Distribution

We first measured how landmarks are visited by mobile nodes. Due to page limit, we only show the visiting distribution of the 5 most visited landmarks in the two traces in Figure 6.1(a) and Figure 6.1(b), respectively. We see that in both traces, for each of the top 5 landmarks, only a small portion of nodes visit it frequently. For example, in the DART trace, less than 15 out of 320 nodes visit landmarks frequently. Though not shown in the figures, such a finding holds for almost all landmarks in the two traces. Thus, we obtain the first observation (O):

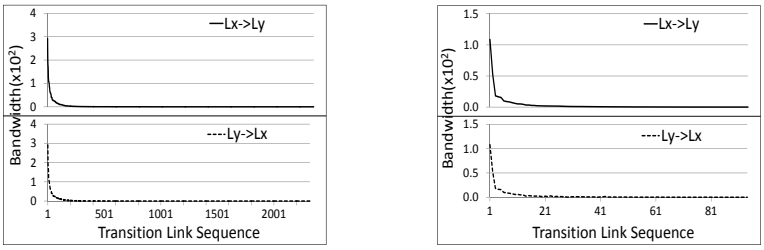
O1: *For each sub-area, only a small portion of nodes visit it frequently.*

This observation matches our daily experience that a department building in a campus usually is mainly visited by students in the department, and a bus station may only be visited by buses that stop at it. Such a finding validates our claim in the introduction section that the number of nodes frequently visiting the destination area is limited, which leads to degraded throughput in previous routing algorithms that only rely on such nodes for packet forwarding.



(a) DART. (b) DNET.

Figure 6.1: Visiting distribution of top 5 most visited landmarks.



(a) DART. (b) DNET.

Figure 6.2: Bandwidth distribution of transit links.

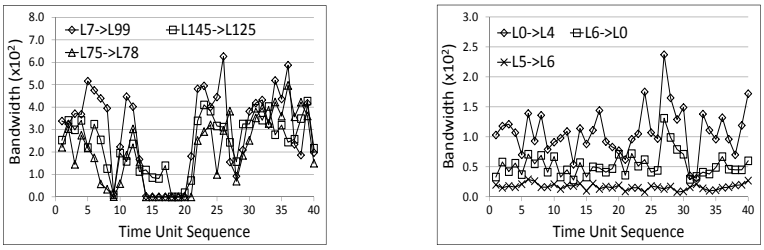
6.1.2.3 Transits among Landmarks

We refer to two transit links containing the same landmarks but have different directions (e.g., T_{ij} and T_{ji}) as *matching transit links*. We then measured the bandwidths of all transit links in the two traces and ordered them in decreasing order. We label two matching transit links with the same sequence number and plot them in two separated sub-figures, as shown in Figure 6.2(a) and Figure 6.2(b). Transit links with 0 bandwidth were omitted. From the two figures, we make the following two observations.

O2: *A small portion of transit links have high bandwidth.*

O3: *The matching transit links are symmetric in bandwidth.*

We also measured the bandwidth of all transit links in the two traces along time. The time unit was set to 3 days and 0.5 day in the two traces, respectively. This results in a total of 40 time units for both traces. Due to page limit, we only present the results of the 3 highest bandwidth transit links in Figure 6.3(a) and Figure 6.3(b). Figure 6.3(a) shows that except two time periods [7, 10] and [14, 21], the measured bandwidth of each transit link fluctuates around its average value slightly. We checked the calendar and found that the two periods are the Thanksgiving and Christmas holidays, which means that few students moved around on the campus. In Figure 6.3(b),



(a) DART. (b) DNET.

Figure 6.3: The transit distribution of top 3 highest bandwidth transit links.

we see that the measured bandwidth of each transit link is more stable around its average bandwidth than in the DART trace. This is because that 1) the DNET trace excludes holidays and weekends, and 2) bus mobility is more repetitive over time than human mobility. Also, both figures show that though there are some fluctuations, the bandwidth relationship of the three transit links remains stable. We then derive:

O4: *The bandwidth of a transit link measured in a certain time period can reflect its overall bandwidth.*

6.2 System Design

In this section, we introduce the detailed architecture of our DTN-FLOW system based on above observations. It has four main components: (1) landmark selection and sub-area division, (2) node transit prediction, (3) routing table construction, and (4) packet routing algorithm. Component (1) provides general guidelines to select the location of landmarks and split the MON into sub-areas. Component (2) predicts the next landmark a node is going to visit based on its previous visiting records. Such predictions are used to forward packets and exchange routing tables among landmarks. Component (3) measures the data transfer capacity between each pair of landmarks, based on which the routing table is built to indicate the next hop landmark for each destination landmark and associated estimated delay. With the support of the first two components, component (4) determines the next-hop landmark and the forwarding node in packet routing.

6.2.1 Landmark Selection and Sub-area Division

The landmark selection determines the places to install landmarks. Sub-area division assigns each landmark a sub-area. Both landmark selection and sub-area division are conducted by the network administrator or planner who hopes to utilize the MON for a certain application.

6.2.1.1 Landmark Selection

As aforementioned, we select popular places that are frequently visited by mobile nodes as landmarks. To identify popular places, a simple way is to collect node visiting history and take top N_v most frequently visited places as popular places. Popular places in MONs with social network structures can also be pre-determined based on node mobility pattern. For example, in the DART network, we can easily find popular buildings that students visit frequently: library, department buildings, and dorms. In MONs in rural areas, villages are naturally popular places. In the MONs using animals as mobile nodes for environment monitoring in mountain areas, places with water/food are frequently visited.

The resulted popular places form a candidate landmark list. There may be several popular places in a small area. Thus, not every popular place needs to be a landmark. Then, for every two candidate landmarks with distance less than D_v meters, the one with less visit frequency is removed from the candidate list. Finally, the distance between every two candidate landmarks is larger than D_v meters.

6.2.1.2 Sub-area Division

With the landmarks, we split the entire network into sub-areas. Since the sub-area division only serves the purpose of routing among landmarks, we do not need a method to precisely define the size of each sub-area. Therefore, we follow below rules to generate sub-areas:

- Each sub-area contains only one landmark.
- The area between two landmarks is evenly split to the two sub-areas containing the two landmarks.
- There is no overlap among sub-areas.

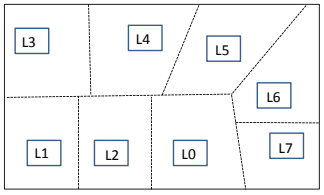


Figure 6.4: Sub-area division in our campus deployment.

Note that the split of area between landmarks does not affect how nodes move between landmarks. Nodes can transit among landmarks through any routes. Figure 6.4 gives an example of the sub-area division in our campus deployment of DTN-FLOW, which is introduced in Section 6.3.3.

6.2.1.3 Influence of Landmark Selection and Sub-area Division

In the above algorithms, N_v and D_v determine the number of landmarks and the sub-area sizes. With more landmarks, a node's transits between landmarks may present higher diversity and may not exhibit a stable pattern for prediction, thereby degrading the routing performance. The maintenance cost of landmarks also increases in this case. With fewer landmarks, the average sub-area size increases, which makes it difficult to provide fine-grained destinations. Therefore, the values of N_v and D_v should be determined so that necessary popular places are represented by landmarks and the patterns of node transits between landmarks can be stably summarized. This objective can be achieved by simply following the above landmark selection and sub-area division process. Recall that landmarks are selected from popular places. Then, a resultant large sub-area with a single landmark is caused by the fact that there are no other popular places in this area. This means even if we place extra landmarks in this area, packets cannot quickly reach them since they are in unpopular places with few node visits. Therefore, N_v and D_v are determined by the popularity of areas, i.e., the node mobility patterns.

6.2.1.4 Real World Scenarios and Limitations

Above landmark selection and sub-area division procedures require certain administration input. However, as previously introduced, this step is quite intuitive and requires slight effort. With the design of landmarks, we can see that DTN-FLOW is suitable for MONs with distributed popular places. In a real-world MON, popular places usually are distributed over an area. For example, mobile device carriers (i.e., people or animals) usually belong to certain social structures and have

skewed and repeated visiting patterns [65]. Therefore, the proposed DTN-FLOW is applicable to most realistic MON scenarios.

6.2.1.5 Cost of Landmarks

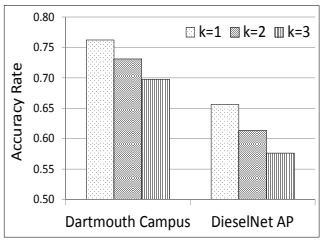
As mentioned previously, a landmark can be regarded as a static autonomous node. Each landmark only needs to communicate with nodes in its sub-area. Therefore, landmarks do not need network connection or to be interconnected. This means that the import of landmarks only needs to build some fixed nodes in the network. Although landmarks require higher capacity in storage and computing compared to normal mobile nodes, these requirements can be easily satisfied on fixed nodes. It is also easy to maintain landmarks. When a landmark malfunctions, we can simply replace it without network-level re-configuration or merge its sub-area with that of a neighboring landmark. Since the number of landmarks often is very limited, the total cost is limited. In summary, the cost of landmark deployment is acceptable, especially when considering the improvement on the routing efficiency and the reduction of the overhead on the mobile nodes.

6.2.2 Node Transit Prediction

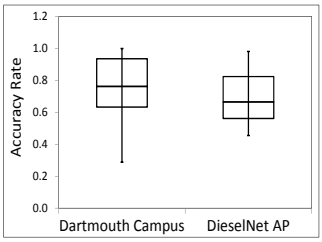
Since DTN-FLOW relies on node transit for packet forwarding, accurate prediction of node transit is a key component. DTN-FLOW predicts each node's next transit by maintaining a landmark visiting history table on each node, as shown in Table 6.2. The "Start time" and "End time" denote the time when a node connects and disconnects to the central station in the corresponding landmark, respectively. Note that the "End time" in previous landmark may differ with the "Start time" in current landmark since a node may not always connect to a landmark during its movement.

Table 6.2: Landmark visiting history table on a node.

Landmark ID	Start time (s)	End time (s)
8	8500	9000
1	7100	8450
7	2000	7000
...	...	...



(a) Average prediction accuracy of order-k predictor.



(b) The minimal, 1st quantile, average, third quantile, and maximal of the accuracy of the order-1 predictor.

Figure 6.5: Accuracy of the transit prediction.

6.2.2.1 Order-k Markov Predictor

To predict node transits among landmarks, we adopt the order-k ($O(k), k = 0, 1, 2, \cdots$) Markov predictor [86], which assumes that the next transit is only related to the past k transits. A node's landmark transit history can be represented by $T_H = T_{x_1,x_2}T_{x_2,x_3}\ldots T_{x_{j-1},x_j}\ldots T_{x_{n-1},x_n}$, in which T_{x_{j-1},x_j} ($x_{j-1} \neq x_j$ and $x_{j-1}, x_j \in [1, M]$) represents a transit from $L_{x_{j-1}}$ to L_{x_j}. We let $X(n-k, n) = T_{x_{n-k},x_{n-k+1}}\cdots T_{x_{n-1},x_n}$ represent the past k consecutive transits. When $k = 0$, $X(n-k, n) = T_{x_n,x_n}$, representing the visiting of landmark L_{x_n}. Then, the probability for each possible next transit $T_{x_n,x_{n+1}}$ of a node is calculated by

$$Pr(T_{x_n,x_{n+1}}|X(n-k,n)) = \frac{Pr(X(n-k,n), T_{x_n,x_{n+1}})}{Pr(X(n-k,n))}, \tag{6.1}$$

where

$$Pr(X(n-k,n), T_{x_n,x_{n+1}}) = Pr(X(n-k,n+1)) \tag{6.2}$$

and

$$Pr(X(n-k,n)) = \frac{N(X(n-k,n))}{N(All_k)} \tag{6.3}$$

where $N(X(\cdot))$ and $N(All_k)$ denote the number of $X(\cdot)$ and k consecutive transits in T_H, respectively. Note that $N(All_0)$ denotes the total number of landmark visits of the node. Then, the transit that leads to the maximal probability based on Equ. (6.1) is selected as the predicted transit. For example, suppose we use an order-1 Markov on a system with 5 landmarks ($M = 5$), and the landmark transit history of a node is $T_H = T_{0,1}T_{1,3}T_{3,4}T_{4,2}T_{2,0}T_{0,1}$. Then, based on Equ. (6.1), the probability for each possible next landmark L_a ($a \in [1, 5]$) is calculated as $\frac{Pr(T_{0,1}T_{1,a})}{Pr(T_{0,1})}$. Based on Equ. (6.3), $Pr(T_{0,1}T_{1,3}) = 1/5$ since $T_{0,1}T_{1,3}$ appears once in T_H and the total number of 2 consecutive transits is 5. Similarly, $Pr(T_{0,1}) = 2/6$. Then, the transit probability is 0.6.

6.2.2.2 Determination of k

In general, the prediction accuracy of the order-k Markov increases as k increases until a certain value due to insufficient position records [69]. The order-k Markov predictor actually exploits the $(k+1)$-hop transit pattern for prediction. Therefore, when k increases, more information is considered to classify node mobility, thus increasing the prediction accuracy. However, when k increases, the possibility that at least one transit in a $(k+1)$-hop transit pattern cannot be collected (i.e., a missed $(k+1)$-hop transit pattern) also increases. When k increases to a large value, too many $(k+1)$-hop transit patterns may be missed, leading to a low prediction accuracy. In other words, the completeness of the collected position information affects the k that can lead to the highest prediction accuracy. Therefore, the administrator needs to first collect nodes' historical visiting records and then check which k can lead to the highest prediction accuracy. The identified k then can be used for future prediction.

6.2.2.3 Prediction Accuracy with the Two Traces

We applied the order-k Markov predictor to both the DART and the DNET traces with k equals to 1, 2, and 3 to check which k can lead to the best prediction accuracy. We calculated the *accuracy rate* of each node as the number of correct predictions over the number of predictions. The average accuracy rates of all nodes with different k are shown in Figure 6.5(a). We see that $k = 1$ leads to the highest prediction accuracy. This is because many position records are absent in the two traces. In DART trace, a student's device cannot be logged unless he/she is using the device. In DNET trace, the APs are roadside APs owned by others and are not dedicated for the experiment, which means they may not appear constantly in the trace, leading to missing records. Based on such a result, we use the order-1 Markov predictor in the experiments in this paper.

We further show the minimal, 1st quantile, average, third quantile and maximal of the accuracy rates of all nodes with the order-1 Markov predictor in Figure 6.5(b). We see that in the DART trace, the accuracy rates of over 75% of nodes are higher than 64%, and the average accuracy rate of all nodes is about 77%. In the DNET trace, the accuracy rates of over 75% of nodes are higher than 59%, and the average accuracy rate of all nodes is about 66%. It is intriguing to see that the prediction accuracy in the bus network in DNET, which should have more repetitive moving patterns, is lower than that in the student network on campus in DART. We believe this is caused

by the reason that we only predict one AP for the next transit while a bus may associate with one of several neighboring APs after each transit in the trace. Though certain inaccurate predictions exist, the routing efficiency can be ensured with a method that will be explained in Section 6.2.4.

6.2.3 Routing Table Construction

In DTN-FLOW, each landmark dynamically measures the bandwidths of its transit links to each neighbor landmark. The bandwidth of a transit link represents the expected delay of forwarding data through it. Based on the estimated delay, each landmark uses the distance-vector method [47] to build a routing table indicating the next hop landmark for each destination landmark. Each landmark periodically transfers its routing table to its neighbor landmarks for routing table update. This step is realized through mobile nodes, i.e., a landmark, say L_i, chooses its node with the highest predicted probability of visiting L_j to forward its routing table to L_j. The detailed processes are introduced below.

6.2.3.1 Transit Link Bandwidth Measurement

Each landmark maintains a bandwidth table as shown in Table 6.3 to record the bandwidth from it to each of its neighbor landmarks. We let N_{ij}^t denote the number of nodes that have moved from L_i to L_j in the t-th time unit. Each landmark, say L_i, periodically updates its bandwidth to landmark L_j by

$$B_{ij_{new}} = \alpha B_{ij_{old}} + (1-\alpha) N_{ij}^t \tag{6.4}$$

in which $B_{ij_{new}}$ and $B_{ij_{old}}$ represent the updated and previous bandwidth, respectively, and α is a weight factor.

Table 6.3: Bandwidth table on a node.

Landmark ID	Measured Bandwidth	Time Unit Sequence
2	20	9
6	6	9
1	15	9
...	...	...

It is easy for landmark L_i to calculate N_{ji}^t since mobile nodes moving to L_i can report their previous landmarks to L_i. However, it is difficult for L_i to calculate N_{ij}^t because after a mobile node moves from L_i to L_j, it cannot communicate with L_i. Recall that **O3** indicates that two matching transit links are symmetric in bandwidths. In this case, L_i can regard $N_{ij}^t \approx N_{ji}^t$ and calculate

$B_{ij_{new}}$ using Equ. (6.4).

However, the symmetric property does not always hold true. For example, transit links connecting two stations in a one way road can hardly be symmetric in bandwidth. To solve this problem, L_i relies on L_j to keep track of N_{ij}. When landmark L_j predicts that a node is going to leave it for L_i, it forwards N_{ij} to the node. When L_i receives N_{ij} from L_j, it checks whether the time unit sequence in it is larger than the current one. If yes, it updates its bandwidth to L_j accordingly based on Equ. (6.4). Otherwise, the packet is discarded.

6.2.3.2 Building Routing Tables

With the bandwidth table, each landmark can deduce the expected delay needed to transfer W bytes of data to each of its neighbor landmarks. Recall T denotes the time unit for B_{ij} measurement. Suppose each node has S bytes of memory, then the expected delay for forwarding a packet from L_i to L_j (D_{ij}) is $D_{ij} = \frac{W}{B_{ij}S}T$. Then, the routing table on each landmark is initialized with the delays to all neighbor landmarks. Each landmark, L_i, further uses the distance-vector protocol to construct the full routing table (as shown in Table 6.4) indicating the next hop for every destination landmark (L_d) in the network and the overall delay from L_i to L_d, denoted by $D(L_i, L_d)$.

Table 6.4: Routing table on one node.

Des. Landmark ID	Next Hop ID	Overall Delay
1	1	7
5	5	3
9	1	18
...	...	...

In the distance-vector protocol, each landmark periodically forwards its routing table and associated time unit to all neighbor landmarks through mobile nodes. When a landmark, say L_i, receives the routing table from a neighbor landmark, say L_j, it first checks whether it is newer than the previously received one. If not, the table is discarded. Otherwise, the routing table is processed one entry by one entry. For each entry, if the destination landmark, L_d, does not exist in the routing table of L_i, it is added to the routing table by setting the "Next Hop ID" as L_j and the "Overall Delay" as $D_{ij} + D(L_j, L_d)$. If L_d already exists, it checks whether $D(L_i, L_d) \leq D_{ij} + D(L_j, L_d)$. If yes, no change is needed. Otherwise, the "Next Hop ID" is replaced as L_j and the "Overall Delay" is updated with $D_{ij} + D(L_j, L_d)$. This process repeats periodically, and each landmark finally learns the next hop to reach each other destination landmark with the minimum overall delay in its routing

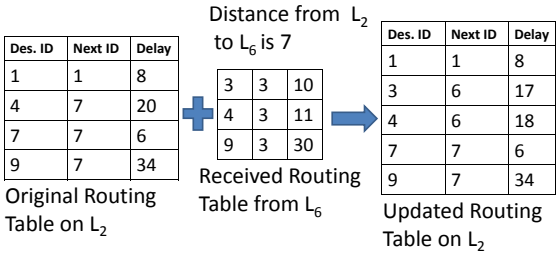


Figure 6.6: Demonstration of the routing table update.

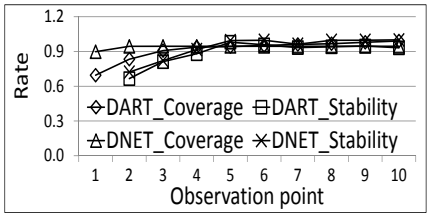


Figure 6.7: Average routing table coverage and stability.

table.

For example, suppose the routing table on L_2 originally contains four entries: (1, 1, 8), (4, 7, 20), (7, 7, 6) and (9, 7, 34), and it receives a routing table from L_6 with 3 entries: (3, 3, 10), (9, 3, 30), (4, 3, 11), and D_{26}=7. Figure 6.6 summaries the routing table update. In detail, since the routing table has no entry for landmark L_3, entry (3, 6, 17) is inserted directly. Though L_9 already exists in the routing table, D_{29}=34 is less than that of relaying through L_6 (i.e., 37), so no change is needed. L_4 already exists in the routing table, and D_{24}=20 is larger than that of relaying through L_6 (i.e., 18), so the "Next Hop ID" is changed to 6 and the "Overall Delay" is set to 18. The final entries in the routing table are (1, 1, 8), (3, 6, 17), (4, 6, 18), (7, 7, 6), and (9, 7, 34).

6.2.3.3 Routing Table Coverage and Stability

We further measured the average coverage and the average stability of all landmarks' routing tables at 10 evenly distributed observation points in the two traces. A landmark's routing table coverage at the i-th observation point is calculated as S_i/N_t, where S_i is the size of its routing table and N_t is the total number of landmarks. A Landmark's stability at the i-th observation point is calculated by 1-E_i/N_t, where E_i is the number of destination landmarks whose next hop landmarks have changed since the previous observation point.

The measurement results are shown in Figure 6.7. We see that after the first several obser-

vation points, each routing table can cover most destination landmarks, which demonstrates that the routing table is capable of providing packet routing guidance to any destinations in our design. We also find that each routing table is quite stable after the first several observation points. This is because node movement presents stable pattern in the two traces, leading to stable bandwidth (i.e. delay) on each link. Therefore, the path to each destination landmark is also stable, leading to stable routing table. This result can be utilized to save the routing table maintenance cost by reducing the routing table update frequency.

6.2.4 Packet Forwarding Algorithm

During the packet forwarding, a landmark refers to its routing table to select the next-hop landmark, and forwards the packet to the mobile node that has the highest predicted probability to transit to the next-hop landmark. However, as mentioned in Section 6.2.2, node transit prediction may not always be accurate, which means a node may fail to carry a packet to the landmark indicated in the routing table. Also, there may be nodes that are moving to the packet's destination node directly, which can be utilized to enhance the routing performance. We first introduce our approaches to handle the two issues and then summarize the routing algorithm.

6.2.4.1 Handling Prediction Inaccuracy

To handle the inaccurate transit prediction, DTN-FLOW follows the principle that every forwarding must reduce the routing latency. Thus, when a node moves from L_i to a landmark L_k other than the predicted one L_j, the node checks whether the new landmark still reduces the expected delay to the destination L_d, that is, whether $D(L_k, L_d) < D(L_i, L_d)$. If yes, the node still forwards the packet to landmark L_k for further forwarding. Otherwise, the node holds the packet, waiting for next landmark that has shorter delay to the destination. This design aims to ensure that each transit, though may not be optimal due to node transit prediction inaccuracy, can always improve the probability of successful delivery.

6.2.4.2 Exploiting Direct Delivery Opportunities

Since nodes move opportunistically in a MON, it is possible that a landmark can discover nodes that are predicted to visit the destination landmarks of some packets. Therefore, when a landmark receives a packet, it first checks whether any connected nodes are predicted to transit to

its destination landmark. If yes, the packet is forwarded to the node directly. In case the node fails to forward the packet to its destination landmark, the node uses the scheme described in Section 6.2.4.1 to decide whether to forward the packet to the new landmark.

6.2.4.3 Routing Algorithm

We present the steps of the routing algorithm as following.

(1) When a node generates a packet for an area, it forwards the packet to the first landmark it meets.

(2) When a landmark, say L_i, generates or receives a packet, it first checks whether any nodes are predicted to move to the destination landmark of the packet. If yes, the packet is forwarded to the node with the highest predicted probability and the expected overall delay, which is used by the carrier node to determine whether to forward the packet to an encountered landmark that is different from the one in the prediction.

(3) Otherwise, L_i checks its routing table to find the next-hop landmark for the packet and inserts the landmark ID and the expected overall delay into the packet.

(4) L_i then checks all connected nodes and forwards the packet to the node that has available memory and has the highest predicted probability to transit to the next-hop landmark indicated by the routing table.

(5) When a node moves to the area of a landmark, say L_j, it forwards L_j all packets that target L_j or have less overall delay from L_j to the destination than L_i. After this, it predicts its next transit based on the order-k Markov predictor and informs this to L_j.

6.2.4.4 Refining Transit Node Selection

In the 4th step of packet routing, the node with the highest predicted probability to transit to the next hop landmark of a packet is selected as its carrier. However, as mentioned in Section 6.2.2, the prediction may not always be correct. We then integrate the prediction accuracy into the process of carrier selection.

Specifically, when selecting the carrier from L_i to L_j, instead of directly using each node's transit probability from L_i to L_j (i.e., $Pr(T_{ij})$), we use an overall transit probability, denoted by

$Pc(T_{ij})$; $Pc(T_{ij}) = Pr(T_{ij}) * Ar(L_i)$, in which $Ar(L_i)$ is a node's prediction accuracy at landmark L_i. It denotes the probability that the node actually transits to the predicted landmark that the node is going to transit to. It is initiated as a medium value (e.g., 0.5), and is multiplied by β ($\beta > 1$) and γ ($\gamma < 1$) when a correct and an incorrect prediction occurs, respectively. Finally, the node with the highest overall transit probability is selected as the carrier.

As a result, the selected carrier should have both high transit probability and stable mobility pattern (i.e., high prediction accuracy) and can improve the probability of carrying the packet to the next hop landmark indicated in the routing table.

6.2.4.5 Communication Scheduling

When a node connects to a landmark, it uses the uplink to upload packets to the landmark. Meanwhile, the landmark utilizes the downlink to forward packets on it to nodes. Both steps follow the packet routing algorithm introduced in Section IV-D3. We assume that the landmark can only communicate with one node through either the uplink or the downlink at a moment. Though nodes usually are sparsely distributed in MONs, a few landmarks may be congested in DTN-FLOW. We then design a scheduling algorithm to improve the overall throughput against the congestion.

(1) The landmark scans its subarea to discover new nodes every T_{sc}, e.g., 1 minute. If found, the landmark allows the new node to use the uplink to register immediately.

(2) Other than the scanning period, the landmark decides to use the uplink or the downlink based on the ratio of the number of packets it holds (N_{pl}) to the number of packets on all nodes (N_{pn}): $R_{ud} = N_{pl}/N_{pn}$. When $R_{ud} < N_{sm}$, the landmark switches to packet uploading mode and selects nodes to use the uplink to upload packets. When $R_{ud} \geq N_{lg}$, it switches to packet forwarding mode again. N_{sm} and N_{lg} can be dynamically adjusted based on system needs, e.g., $N_{sm} = 1$ and $N_{lg} = 3$.

(3) In the packet uploading mode, the uplink is assigned to the node with the most packets that their expected delays to destination landmarks are lower than their remaining TTLs. The node is allowed to upload at most N_{uu} (e.g., 50) packets each time. Such a process repeats until the landmark switches to the packet forwarding mode.

(4) In the packet forwarding mode, the landmark first forwards the packet that 1) has the minimal remaining TTL and 2) its expected delay to its destination landmark is smaller than the

remaining TTL. Such a process repeats until the landmark switches to the packet uploading mode.

Following this manner, packets that need to be handled first to ensure their successful delivery are assigned higher priority on the landmark, thereby improving the overall throughput.

6.2.5 Advanced Extensions and Discussions

In this section, we further propose three strategies to improve the efficiency and robustness of DTN-FLOW.

6.2.5.1 Dead End Prevention

As mentioned previously, a node may carry a packet to an "unexpected" landmark, which means that it fails to transit to the predicted landmark but moves to a wrong landmark. In this case, based on our routing algorithm, this node still is responsible for carrying the packet to the next hop landmark or a suitable landmark. However, this process may lead to a dead end, in which the packet carrier stays in a wrong landmark for a long time. For example, when moving out of landmark L_i, a bus may move to a parking lot or a garage for maintenance. In this case, the packets on the bus have to wait for a long time, leading to a dead end.

We propose a method to detect the dead end based on a node's historical average stay time in landmarks. In this method, each node calculates and stores the average time it stays in each landmark based on its historical movement records. When a node transits to a landmark, say L_e, it checks if a dead end occurs based on following conditions, where H_t is a determination factor and is usually set to a relatively large value to prevent false positives.

- If it stays in L_e for H_t times longer than its average stay time in a landmark.
- If it has stayed in L_e for H_t times longer than its average stay time in L_e.

The first condition means that the node encounters a dead end on its regular route, while the second condition means it encounters an abrupt dead end, i.e., unexpected maintenance. When a node observes a dead end when it moves to L_e, rather than keeping its packets, it forwards them to L_e directly. Then, L_e utilizes its routing table to decide the next hop landmark for these packets and forward them to the nodes that can carry them out of L_e. Note that in order to reduce false

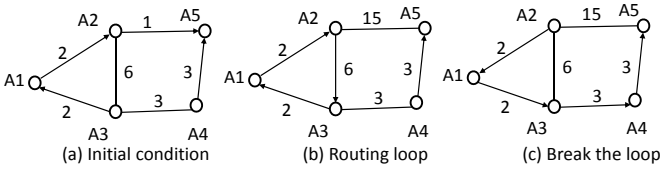


Figure 6.8: Demonstration of the routing loop detection and correction.

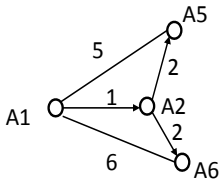
positives, dead end detection is launched only when a node has accumulated enough historical records to calculate its average stay time in each landmark.

6.2.5.2 Routing Loop Detection and Correction

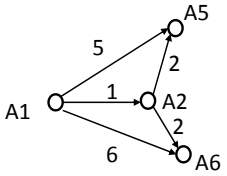
Recall that we use the distance-vector protocol to build the routing tables on each landmark to indicate the next hop landmark to each destination landmark. Specifically, landmarks exchange their routing tables periodically or when necessary, and update their routing tables accordingly, as explained in Section 6.2.3.2.

However, due to untimely routing table updates, routing loop may happen. Figure 6.8(a) and Figure 6.8(b) demonstrate an example of the routing loop regarding the destination landmark $A5$. In Figure 6.8, the number on each link denotes the expected delay of the link. As shown in Figure 6.8(a), initially, the next hop landmark for A_5 in the routing tables on $A1$, $A2$, $A3$, and $A4$ are $A2$, $A5$, $A1$, and $A5$, respectively. Then, as shown in Figure 6.8(b), suppose the delay of the link connecting $A2$ and $A5$ changes to 15, and this information is only known by $A2$. Meanwhile, a distance vector from $A3$ arrives at $A2$ claiming $A3$'s estimated delay to $A5$ is 5 (i.e., 2+2+1) through $A1$. In this case, $A2$ will change the next hop landmark for $A5$ from $A5$ to $A3$, which leads to a routing loop of $A2 \rightarrow A3 \rightarrow A1 \rightarrow A2$ for packets targeting $A5$.

In order to detect and correct routing loops, we let each packet record the IDs of the landmarks it has visited. When a packet finds that it has visited a landmark twice, it informs this landmark the existence of a routing loop and the involved landmarks in the loop (e.g., $A1$, $A2$, and $A3$ in Figure 6.8(b)). Then, the landmark generates a loop correction packet, which includes the IDs of the involved landmarks and the destination landmark, and sends it to all involved landmarks. Upon receiving such a correction packet, these landmarks immediately send its updated distance vector on the destination landmark to all neighbor landmarks repeatedly until the next hop landmark for the destination landmark remains unchanged for a certain period of time T_l. T_l should be large enough so that each landmark in the loop can collect the updated distance vector from all other



(a) Packets from $A1$ destined to $A2$, $A5$, and $A6$ overload link T_{12}.



(b) Link T_{15} and T_{16} alleviate the load on link T_{12}

Figure 6.9: Demonstration of an overloaded link and solution.

landmarks in the loop. We then set T_l to the average time for a packet to traverse the loop.

6.2.5.3 Load Balancing

In above design, DTN-FLOW decides the next hop landmark by solely considering the delay of the links, i.e., choosing the link that leads to the destination landmark with the minimal expected delay. However, such a strategy may generate overloaded links because a link with a very low expected delay may be included in the optimal routes to multiple destination landmarks. Figure 6.9(a) shows such an example, in which the number on each link denotes its expected delay. In the figure, the packets generated on $A1$ destined to $A2$, $A5$ and $A6$ will select the transit link T_{12} since it has a very low expected delay, (i.e., 1), thereby possibly overloading the link T_{12}.

When a link is overloaded, the packets may wait for a long time to be forwarded through the link. They may take much longer time than the expected delay of the link to pass through the link, thereby degrading the routing efficiency. Actually, when many packets wait for the same link, other links can be utilized to offload the load. For example, in Figure 6.9(b), the transit link T_{15} and $T16$ can take some packets from $A1$ destined to $A5$ and $A6$ to reduce the load on link T_{12}.

For this purpose, we first expand the routing table to provide a backup next hop landmark for each destination landmark, which has the second lowest overall delay to the destination landmark, as shown in Table 6.5. The backup next hop landmark is updated concurrently with the update of the original routing table, which does not incur additional communication cost. Then, each landmark monitors the incoming rate and outgoing rate for each link. The former is calculated as the average number of received packets that need to be forwarded through the link in a time unit, while the latter is calculated as the average number of packets that are carried by mobile nodes to pass through the link in a time unit. When the incoming packet rate is T_b times larger than the

outgoing rate, it means that the number of received packets increases faster than the number of packets being forwarded out through the link, which leads to link overloaded. Then, the landmark forwards the packets to the backup next hop landmark through another link.

Table 6.5: Expanded routing table in one node.

Des. Landmark	Next Hop	Delay	Backup	Delay
1	1	7	14	20
5	5	3	11	8
9	1	18	7	30
...	...	...	...	...

6.2.5.4 Routing Packets to Mobile Nodes

Recall that DTN-FLOW is mainly designed to realize packet routing between different sub-areas/landmarks. In certain scenarios, it is also desirable to route a packet to a certain mobile node. DTN-FLOW can be adapted to realize this objective. In MONs, mobile nodes usually have skewed visiting preferences [65], which means that they visit certain landmarks frequently. This property can be utilized to forward a packet to a mobile node efficiently. Nodes can summarize their most frequently visited landmarks and report such information to landmarks in the network. Thus, the sender of a packet destined to a destination node can first learn the destination's frequently visited landmarks and forward/copy the packet to them. Since the destination node visits these landmarks frequently, the packet is unlikely to stay in the landmarks for a long time before being forwarded to the destination. This scheme avoids chasing the mobile nodes continuously or the requirement of knowing the position of the destination node beforehand.

6.3 Performance Evaluation

We first conducted trace-driven experiments with both the DART and the DNET traces and then evaluated the extensions introduced in Section 6.2.5. A small DTN-FLOW system is also deployed on our campus.

6.3.1 Trace-driven Experiments

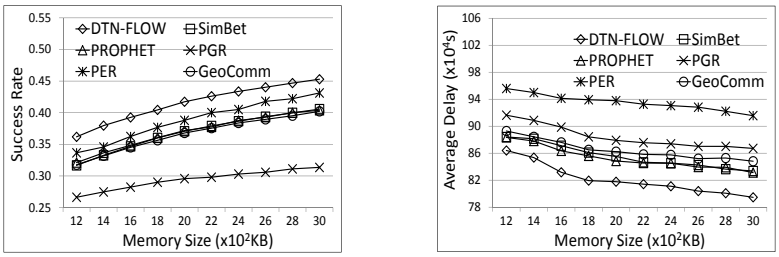
6.3.1.1 Experiment Settings

We used the first 1/4 part of the two traces as the initialization phase, in which nodes construct routing tables. Then, packets were generated at the rate of R_p packets per landmark per day. R_p was set to 500 by default. The destination landmark of each packet is randomly selected. We set the TTL (Time to Live) of packets to 20 days in the DART trace and 4 days in the DNET trace. A packet is dropped after its TTL expires. The time unit T for bandwidth evaluation and routing table update was set to 3 days. The size of each packet was set to 1KB, and each node's memory was set to 2000KB by default. The memory of the landmark was not limited. We used the order-1 Markov predictor in the experiments.

We compared DTN-FLOW with five state-of-the-art MON routing algorithms: SimBet [30], PROPHET [72], PGR [62], GeoComm [36], and PER [93]. They were originally proposed for node-to-node routing or packet dissemination in MONs. We adapted them to fit landmark-to-landmark routing to make them comparable to DTN-FLOW. We use SimBet to represent the social network based routing methods. It combines centrality and similarity to calculate the suitability of a node to carry packets to a given destination landmark. The similarity is derived from the frequency at which the node visits the landmark. We use PROPHET to represent the probabilistic routing methods. It simply employs the visiting records with landmarks to calculate the future meeting probability to guide the packet forwarding. PGR, GeoComm, and PER exploit geographical information for MON routing. PGR uses observed node mobility routes, i.e., a sequence of locations, to check whether the destination landmark is on a node's route. GeoComm measures each node's contact probability per unit time with each geo-community, i.e., landmark, to guide the packet routing. In PER, a node's past mobility and sojourn among different landmarks are summarized to provide prediction a node's probability to visit a landmark before a certain deadline.

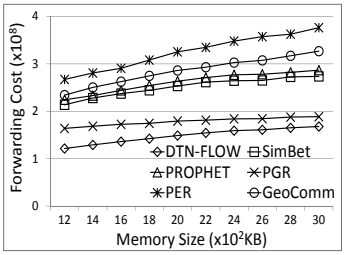
We measured following metrics.

- *Success rate*: The percentage of packets that successfully arrive at their destination landmarks.
- *Average delay*: The average time per successfully delivered packet needed to reach the destination landmark.
- *Forwarding cost*: The number of packet forwarding operations occurred during the experiment.

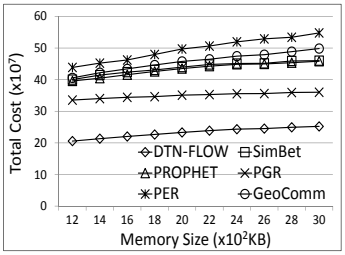


(a) Success rate.

(b) Average delay.



(c) Forwarding cost.



(d) Total cost.

Figure 6.10: Performance with different memory sizes using the DART trace.

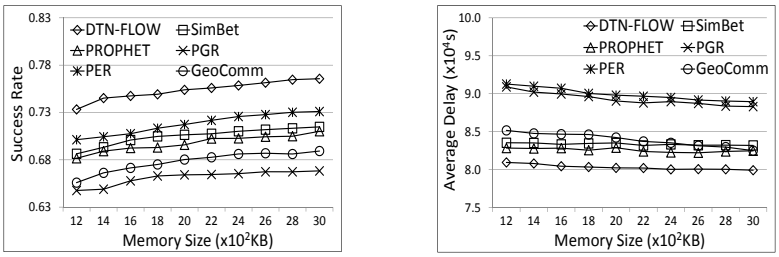
- *Overall cost*: The total number of packet and routing information forwarding operations during the experiment. The cost of forwarding a routing table or a meeting probability table with m entries is counted as m.

Recall that in DTN-FLOW, landmark deployment is completed off-line before the system starts. Thus DTN-FLOW incurs additional cost for landmark deployment compared to other algorithms. However, this small additional cost can improve the routing efficiency and reduce the forwarding costs of energy-constraint mobile nodes (as shown in the experimental results later on), which is the key advantage of DTN-FLOW.

6.3.1.2 Performance with Different Memory Sizes

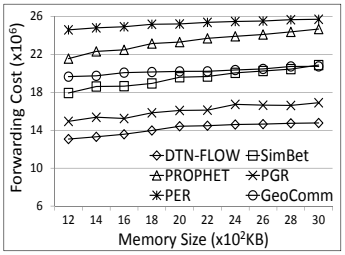
We first evaluated the performance of the six methods when the size of memory in each node was varied from 1200KB to 3000KB with a 200KB increase in each step.

Success Rate: Figure 6.10(a) and Figure 6.11(a) present the success rates of the six methods with the DART and the DNET traces, respectively. We see that when the memory in each node increases, the success rates always follow DTN-FLOW>PER>SimBet≈PROPHET>GeoComm>PGR. DTN-FLOW has the highest success rate because it fully utilizes node movements to forward packets one landmark by one landmark to their destination landmarks, even though some nodes rarely

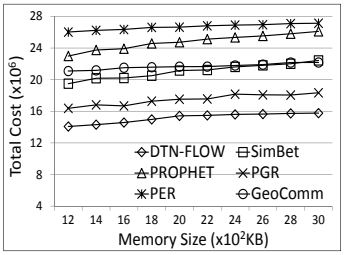


(a) Success rate.

(b) Average delay.



(c) Forwarding cost.



(d) Total cost.

Figure 6.11: Performance with different memory sizes using the DNET trace.

or may not visit these destinations. On the contrary, other methods only rely on nodes that visit destinations frequently for packet forwarding. Limited number of such nodes prevents them from achieving high success rate.

PER leads to the second highest success rate because it considers both transit probability distribution and sojourn time probability distribution to predict a node's probability to move to a landmark within a time limit. SimBet and PROPHET present similar success rates. SimBet exploits social properties, i.e., centrality and similarity, to rank a node's suitability to carry packets to a landmark. PROPHET uses previous encountering records to predict a node's probability of visiting a landmark. Both metrics can indirectly reflect a node's ability to visit a landmark, leading to high and similar success rates.

GeoComm has similar success rate with and lower success rate than SimBet and PROPHET in the tests with DART trace and the DNET trace, respectively. This is because in the DNET trace, each node, i.e., bus, has even contact probability with landmarks on its route since it stays on these landmarks with equal amount of times. Then, the contact probability cannot reflect a node's probability to visit landmarks as accurate as in SimBet and PROPHET, leading to lower success rate.

PGR tries to predict the entire route of a node (with multiple landmarks) for packet for-

warding. However, such a prediction has a low accuracy. As shown in Figure 6.5(b), the average accuracy of the prediction of only one location is already below 80%. Therefore, PGR has the lowest success rate.

In summary, the experimental results verify the high throughput of DTN-FLOW in transferring data among landmarks with difference memory sizes on each node.

Average Delay: Figure 6.10(b) and Figure 6.11(b) show the average delays of successfully delivered packets in the six methods with the DART and the DNET traces, respectively. We see that the average delays follow DTN-FLOW<SimBet≈PROPHET<GeoComm<PER<PGR. DTN-FLOW has the lowest average delay because the designed routing tables in landmarks guide packets to be forwarded along the fastest paths to their destinations. In SimBet and PROPHET, packets may be generated in or carried to areas where very few nodes move to their destinations regularly. Therefore, packets have to wait for a certain period of time before meeting nodes that visit their destinations frequently, leading to a moderate average delay. Moreover, since nodes with high centrality (i.e., connecting many landmarks) may not visit the specific destination landmark as frequently, SimBet has slightly higher average delay than PROPHET.

For GeoComm, the contact probability between a landmark and a node cannot reflect the node's future probability to visit a landmark as accurate as that in SimBet and PROPHET. Therefore, it has larger average delay than SimBet and PROPHET. PER further has larger delay than GeoComm because it only chooses the node that has the highest probability to visit the destination landmark before a deadline as the forwarder for a packet, rather than the node that can carry the packet to the destination landmark as soon as possible. For PGR, as explained previously, it is difficult to accurately predict long paths with multiple locations, thus leading to inaccurate forwarder selection and the long delay.

These experimental results show the high efficiency of DTN-FLOW in transferring data among landmarks with difference sizes of memory in each node.

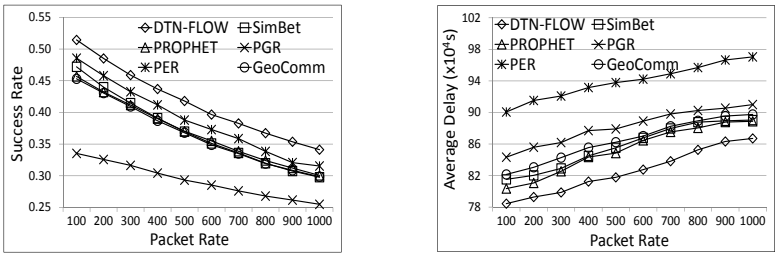
Forwarding Cost: Figure 6.10(c) and Figure 6.11(c) plot the forwarding costs of the six methods with the DART and the DNET traces, respectively. We find that the forwarding costs follow DTN-FLOW<PGR<SimBet<PROPHET<PER with both traces and GeoComm has the second and the third largest forwarding cost with the DART and the DNET trace, respectively. DTN-FLOW refers to the routing table to forward packets along fastest landmark paths to reach their destinations, which usually takes several forwarding operations.

PGR has the second lowest forwarding cost because nodes tend to show similar ability to visit a landmark. Therefore, a packet holder cannot easily find another node that has higher probability of meeting the destination node. Then, packets are not forwarded frequently. However, the low forwarding cost in PGR also results in a low efficiency.

SimBet, PROPHET, GeoComm, and PER use a metric to rank the suitability of nodes for carrying packets and forward packets to high rank nodes. Then, packets are frequently forwarded to nodes with higher suitability, leading to high forwarding cost. More specifically, the easiness of finding a node with higher rank determines the actual forwarding costs of the four methods. In SimBet, since high-centrality nodes usually are limited in the network, packets are gathered on these nodes without further forwarding, leading to the lowest forwarding cost among the four methods.

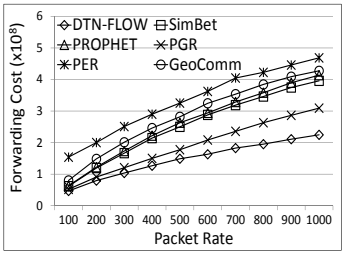
GeoComm has higher and lower forwarding cost than PROPHET in the test with the DART trace and the DNET trace, respectively. This is because in GeoComm, a node's contact probabilities with each landmark vary greatly due to people's mobility in the DART trace and remain stable in the DNET trace. Therefore, packets are frequently forwarded in the test with DART trace. On the contrary, PROPHET forwards packets greedily by only considering meeting frequency, leading to high forwarding cost in both traces. PER leads to the highest forwarding cost in the tests with both traces. This because whenever a node moves to a new landmark, its probability of visiting a certain landmark before a deadline changes. In other words, such probabilities vary significantly with node movement. As a result, packets are forwarded for the most frequently in the network.

Total Cost: Figure 6.10(d) and Figure 6.11(d) plot the total costs of the six methods with the DART and the DNET traces, respectively. We see that the total costs follow DTN-FLOW <PGR<SimBet≈PROPHET<GeoComm<PGR in the tests with the DART trace and DTN-FLOW<PGR<SimBet≈GeoComm<PROPHET<PGR in the tests with the DNET traces. Recall that the total cost includes packet forwarding cost and maintenance cost, which is incurred by routing information forwarding. In DTN-FLOW, the maintenance cost comes from routing table updates. When a node connects to a new landmark, it forwards the routing table of its previously connected landmark to the new landmark and receives the routing table of the new landmark. In other methods, two encountering nodes exchange their calculated suitability/rank for each destination landmark and then decide whether to forward packets to the other node. Since a node's probability of meeting a landmark is lower than that of meeting another node, maintenance cost in DTN-FLOW is lower than that in other methods. Therefore, DTN-FLOW produces the lowest total cost.

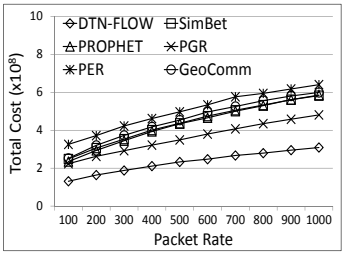


(a) Success rate.

(b) Average delay.



(c) Forwarding cost.



(d) Total cost.

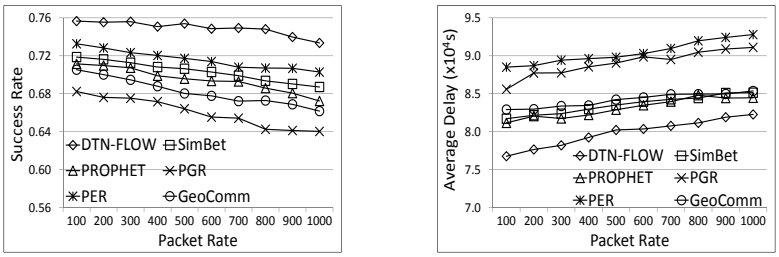
Figure 6.12: Performance with different packet rates using the DART trace.

Comparing Figure 6.10(d) and Figure 6.11(d) with Figure 6.10(c) and Figure 6.11(c), we notice the maintenance cost only counts a small part of the total cost. Therefore, the relationship on total cost remains the same as that on the forwarding cost. We also see that when the memory size on each node increases, the total costs of all methods increase, though the maintenance costs of each method actually remain stable. This is because the forwarding cost is much higher than the maintenance cost. The results on forwarding cost and total cost verify the high efficiency of DTN-FLOW in terms of cost with different memory sizes on each node.

6.3.1.3 Performance with Different Packet Rates

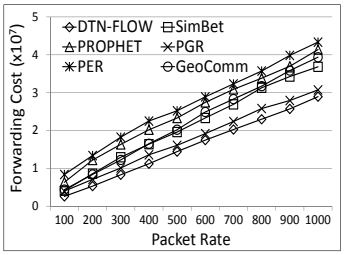
We also evaluated the performance of the six methods with different packet generation rates. We varied the packet rate from 100 to 1000 with 100 increase in each step.

Success Rate: Figure 6.12(a) and Figure 6.13(a) show the success rates of the six methods in the tests using the DART and the DNET traces, respectively. We see that the success rates follow DTN-FLOW>PER>SimBet≈ PROPHET>GeoComm>PGR. Such results match those in Figure 6.10(a) and Figure 6.11(a) for the same reasons. We also see that when the packet rate increases, the success rates of the six methods decrease. The forwarding opportunities in the system are determined by node memory and encountering opportunities, which are independent with the

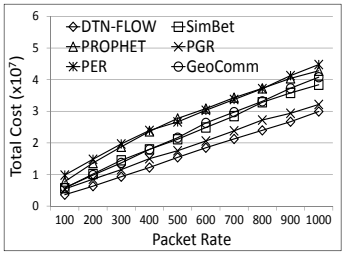


(a) Success rate.

(b) Average delay.



(c) Forwarding cost.



(d) Total cost.

Figure 6.13: Performance with different packet rates using the DNET trace.

number of packets. When the number of packets increases, the number of packets that can be delivered successfully does not increase accordingly, leading to a degraded success rate. The high success rate of DTN-FLOW with different packet rates verifies the high throughput performance of DTN-FLOW.

Average Delay: Figure 6.12(b) and Figure 6.13(b) illustrate the average delays of the six methods in the tests using the DART and the DNET traces, respectively. We see that the average delays follow DTN-FLOW<SimBet≈PROPHET <GeoComm<PGR<PER. This relationship remains the same as in Figure 6.10(b) and Figure 6.11(b) for the same reasons. Moreover, we find that when the packet rate increases, the average delays of the four methods increase. This is caused by the limited forwarding opportunities in the system. When there are more packets in the system, the average time a packet needs to wait before being forwarded increases, resulting in higher total delay. DTN-FLOW always generates the lowest average delay at all packets rates, which demonstrates the high efficiency of DTN-FLOW in terms of routing delay.

Forwarding Cost: Figure 6.12(c) and Figure 6.13(c) show the forwarding costs of the six methods in the tests using the DART and the DNET traces, respectively. We see that the forwarding costs DTN-FLOW<PGR<SimBet <PROPHET<PER with both traces and GeoComm has the second and the roughly third largest forwarding cost with the DART and the DNET trace,

respectively. Again, this relationship is the same as in Figure 6.10(c) and Figure 6.11(c) due to the same reasons. We also see that the forwarding costs of the four methods increase when the packet rate increases. When there are more packets generated in the system, more forwarding opportunities are utilized, resulting in more packets forwarding operations. This is why the forwarding costs in Figure 6.13(c) remain relative stable when the packet rate is larger than 40.

Total Cost: Figure 6.12(d) and Figure 6.13(d) show the total costs of the six methods in the tests using the DART and the DNET traces, respectively. We see that their total costs again follow DTN-FLOW<PGR<SimBet≈PROPHET< GeoComm<PER and DTN-FLOW<PGR< SimBet ≈ GeoComm < PROPHET < PER in the tests with the DART and the DNET traces, respectively. This result matches that in Figure 6.10(d) and Figure 6.11(d) for the same reasons. We also find that the total costs of the four methods increase when the packet rates increase. This is because that the maintenance costs of the four methods, which are irrelevant to the packet rate, only account for a small part of the total costs. Such results further confirm the high efficiency of DTN-FLOW in terms of cost with difference packet rates.

Combining all above results obtained with various memory sizes and packet rates, we conclude that DTN-FLOW has superior performance in achieving high throughput, low average delay, and low cost data transmission between landmarks than previous routing algorithms in MONs.

6.3.2 Evaluation of Advanced Algorithm Extensions

In this section, we evaluate the performance of the extensions proposed in Section 6.2.5. We set the packet rate to 500 and the memory size on each node to 2000 KB. Other settings are the same as in the trace-driven experiments.

6.3.2.1 Dead End Prevention

We evaluated the performance of our proposed dead end prevention method. We varied H_t from 2 to 5 in the test. Table 6.6 shows the hit rates and average delays of each test. Note the "ORG" represents the original DTN-FLOW without the proposed dead end prevention method.

We see from the table that in the tests with both traces, when the dead end prevention method is used, the success rate is increased and the average delay is decreased. This result confirms that our proposed method enables nodes to effectively detect dead ends and transfer their packets to other nodes through landmarks. We also find that the best performance is achieved when H_t

Table 6.6: Experimental results on dead end prevention.

Trace \ H_t		ORG	2	3	4	5
DART	Success Rate	0.410	0.433	0.431	0.429	0.428
	Delay ($\times 10^5 S$)	8.19	7.89	7.94	7.99	8.01
DNET	Success Rate	0.747	0.761	0.760	0.751	0.749
	Delay ($\times 10^4 S$)	8.02	7.46	7.49	7.55	7.58

equals to 2 in the tests with both traces. This shows that $H_t = 2$ is sufficient to detect most dead ends. When H_t is larger than 2, it requires more time to identify a dead end. Then, some packets maybe dropped due to TTL during the waiting, leading to more dropped packets (i.e., decreased success rate) and increased average delay.

6.3.2.2 Routing Loop Detection and Correction

We also evaluated the effectiveness of the loop detection and correction method proposed in Section 6.2.5.2. We purposely created L_s loops in this test and tested when L_s equals to 2 and 3. The destination landmark of each created loop is randomly selected in the network. Table 6.7 shows the experimental results with both traces, in which "ORG-x" and "W-x" (x=2, 3) represent the DTN-FLOW without and with the proposed loop detection and prevention method when L_s equals x, respectively.

Table 6.7: Experimental results on loop detection and correction.

Trace \ L_s		ORG-2	W-2	ORG-3	W-3
DART	Success Rate	0.379	0.404	0.362	0.373
	O. Delay ($\times 10^6 S$)	6.44	6.07	6.49	6.02
DNET	Success Rate	0.738	0.745	0.732	0.740
	O. Delay ($\times 10^5 S$)	5.47	5.12	5.51	5.18

We see from the table that when 2 or 3 routing loops exist in the network, the hit rates decrease in the tests with both traces without the proposed loop detection and correction method. This is because some packets are continuously forwarded along the loop, failing to reach their destinations. We also find that the hit rates in W-2 and W-3 are only slightly lower than those when there are no routing loops as shown in Figures 6.12(a) and 6.13(a). Such a result demonstrates that our proposed method can effectively detect and correct loops.

In order to compare the delay fairly, we measure the overall average delay, denoted O. Delay, in this test, which calculates the average delay of all packets (including the unsuccessful packets). We regard the delay of an unsuccessful packet as the experimental time, i.e., 10^7 seconds for the DART trace and 2×10^6 seconds for DNET trace. We see from the table that when the loop detection

correction method is used, the overall average delay is decreased. This is because the proposed method can reduce the number of unsuccessful packets due to routing loops, thereby decreasing the overall average delay.

6.3.2.3 Load Balancing

We further evaluated the performance of the proposed load balancing method. For the success rate, in order to better demonstrate the effectiveness of our proposed method, we purposely enlarged the packet rate to the range of [1100, 1500] to create overloaded links in the network.

Table 6.8: Experimental results of load balancing on success rate.

Packet Rate ($\times 10^2$)		11	12	13	14	15
DART	W/O-Balance	0.315	0.307	0.295	0.283	0.260
	W-Balance	0.319	0.313	0.307	0.297	0.280
DNET	W/O-Balance	0.679	0.645	0.610	0.584	0.541
	W-Balance	0.696	0.663	0.628	0.603	0.568

Table 6.9: Experimental results of load balancing on average delay.

Packet Rate ($\times 10^2$)		11	12	13	14	15
DART($\times 10^4$s)	W/O-Balance	89.4	89.7	89.8	90.3	90.8
	W-Balance	87.6	87.4	87.0	87.2	87.5
DNET($\times 10^4$s)	W/O-Balance	8.44	8.62	8.71	8.79	8.94
	W-Balance	8.36	8.43	8.66	8.71	8.81

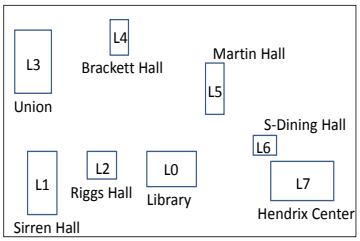
Table 6.8 and 6.9 show the experimental results on hit rates and average delays of DTN-FLOW with and without the load balancing method, denoted by "W-Balance" and "W/O-Balance", respectively. We see from the two tables that when the load balancing method is used, the success rate is increased and the average delay is decreased in the tests with both traces. This is because the backup next hop landmark effectively offload packets waiting for overloaded links, thereby reducing their waiting time. These results show that the proposed load balancing method can effectively offload packets on overloaded links to improve the overall routing efficiency.

6.3.3 Real Deployment

We deployed DTN-FLOW on our campus for real-world evaluation of its performance.

6.3.3.1 Settings

We selected 8 buildings as landmarks and labeled them as L_0 to L_7. Their relative locations are shown in Figure 6.14(a). Among the 8 landmarks, L_0 is the library, L_1, L_2, L_4, and L_5 are

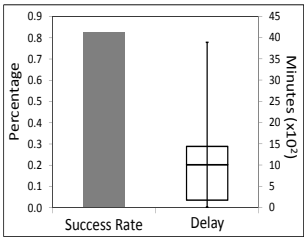


(a) Map for landmark locations.

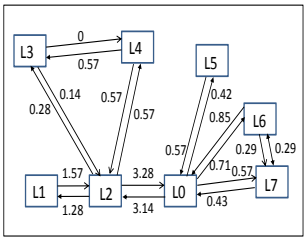
Item	Value
# Landmarks	8
# Nodes	9
# Transits	147
# Packets	2100
Duration	4 days
Packet size	1 KB
Node memory	50 KB

(b) Configuration

Figure 6.14: Landmark map and configurations in the real deployment.



(a) Success rate and delay.



(b) Bandwidths of transit links

Figure 6.15: Experimental results in real deployment.

department buildings, and L_3, L_6, and L_7 are the student center and dining halls. Each of 9 students from 4 departments carried a Windows Mobile phone daily, each of which checks its GPS coordinator periodically to judge the landmark it associates with.

In the test, each landmark generates 75 packets evenly in the daytime each day. We simulated a scenario in which L_0 (Library) needs to collect information from other buildings, i.e., all packets were targeted to L_0. The packet TTL was set to 3 days. We set the packet size to 1KB and the memory on each node to 50KB. The time unit T was set to 12 hours. The deployment configuration is summarized in Figure 6.14(b).

6.3.3.2 Experimental Results

Figure 6.15(a) demonstrates the success rate and the minimal, first quantile, average, third quantile, and maximal of the delays of successfully delivered packets. We see that more than 82% of packets were successfully delivered to the destination. Also, more than 75% of packets were delivered within 1400 minutes, and the average delay is about 1000 minutes. Note that the entire deployment only employed 9 mobile nodes with 147 transits to forward packets. A larger deployment with more nodes would increase the success rate and reduce the delay. These experimental results demonstrate

the high efficiency of the DTN-FLOW in transferring data among landmarks.

We also obtained the bandwidth of each transit link at the end of the deployment, as shown in Figure 6.15(b). We omit transit links with bandwidth lower than 0.14 to show the major routing paths. The bandwidth on different transit links are within our expectation. For example, the links between L_0 and L_2 have very high bandwidth. This is because most students attended the test are from departments located in L_2 and L_1, and they usually study in the library (L_0) and go to classes in both department buildings (L_1 and L_2). Such results justify that the DTN-FLOW can accurately measure the amount of transits among landmarks.

We further recorded the routing table on each landmark. Due to page limit, we only show those of L_2, L_4, and L_6 in Table 6.10. We see that the routing tables match the fastest path based on transit link bandwidths shown in Figure 6.15(b). For L_2, it needs to go through L_0 to reach L_0, L_5, L_6, L_7. For L_4, it relies on L_3 and L_2 to reach other landmarks. For L_6, except for L_7, it has to go through L_0 to reach other landmarks. Such results verify that the routing table update in DTN-FLOW, which relies on mobile nodes, is reliable and can reflect the suitable paths to each destination.

Table 6.10: Routing tables in L_2, L_4, and L_6.

Landmark ID	Destination Landmark	Next-hop
L_2	L_0, L_5, L_6, L_7	L_0
	L_1	L_1
	L_3	L_3
	L_4	L_4
L_4	L_0, L_1, L_3, L_6, L_7	L_3
	L_2, L_5	L_2
L_6	$L_0, L_1, L_2, L_3, L_4, L_5, L_6$	L_0
	L_7	L_7

6.4 Summary

In this chaper, we propose DTN-FLOW, an efficient routing algorithm to transfer data among landmarks with a high throughput in MONs. DTN-FLOW splits the entire MON area into sub-areas with different landmarks, and uses node transits between landmarks to forward packets one landmark by one landmark to reach their destinations. Specifically, DTN-FLOW consists of four components: landmark selection and sub-area division, node transit prediction, routing table construction, and packet routing algorithm. The first component selects landmarks from places that

are frequently visited by nodes and split the network into sub-areas. The second component predicts node transits among landmarks based on previous movement using the order-k Markov predictor. The third component measures the transmission capability between each pair of landmarks to build routing tables on each landmark. In the fourth component, each landmark decides the next-hop landmark for each packet by checking its routing table and forwards the packet to the node that is most likely to transit to the landmark. Extensive analysis, experiments, and real deployment on our campus demonstrate the effectiveness of DTN-FLOW.

Chapter 7

Conclusions and Future Work

Mobile opportunistic networks (MONs) have attracted significant attention recently due to the increasing popularity of mobile devices such as smartphones and mobile sensors. In such a network, mobile devices rely on the intermittent encountering for packet exchange without the support of infrastructures, i.e., peer-to-peer communication. Therefore, MONs can exploit the otherwise wasted communication opportunities resulted from device mobility and bring about many beneficial services. For example, due to the distributed network structure, i.e., no central station is needed, it can enable certain communication services in areas without infrastructures, e.g., rural and mountain areas. Further, since nodes communicate with each other during the encountering, it can enable the encountering based social network services even in areas with communication infrastructures, e.g., campus.

However, due to the aforementioned properties of MONs, file sharing and packet routing in MONs, which are the key for many services built upon MONs, are non-trivial. Therefore, in this dissertation, we instigated how to realize efficient file sharing and packet routing in MONs. Firstly, in Chapter 3, we explore how to create replicas for globally optimal file replication in the scenario when a file requester relies on the encountering with the file holder to retrieve the requested file. In this work, we consider both the ability to meet nodes and storage as available resources. We theoretically analyze the relationship between resource allocation and average file access delay to deduce an optimal file replication rule. Based on the rule, we design the Priority Competition and Split replication protocol (PCS) that can approximate the optimal file replication in a fully distributed manner. Extensive experiments with both real traces and synthesized node mobility

demonstrate the correctness of the optimal file replication rule and the efficiency of PCS.

We further consider the scenario in which a file request can be actively forwarded in the network to locate the requested file in Chapter 4. In this chapter, we consider both device holders' file sharing interests and their contact frequencies to assist the file searching process. We propose a Social network based P2P cOntent file sharing system in mobile Opportunistic Networks (SPOON). We first extract node interests from files on it and then cluster common-interest nodes with frequent contacts into one community. We further exploit node mobility patterns to design nodes that have tight connection with community members for intra-community file searching and highly mobile nodes to bridge requests among communities. Finally, file requests are always forwarded to nodes that are more likely to meet the file holders based on interest similarity and contact frequency. Both GENI testbed experiments and trace-driven simulations prove the efficiency of SPOON.

In addition to file sharing, we also develop an efficient and lightweight MON packet routing algorithm that exploits the social maps on mobile nodes, denoted by SMART, in Chapter 5. SMART is designed for packet routing among nodes in MONs. It exploits the social network property that the frequently met friends of a person often remain stable. Nodes exchange the top L most frequently encountered nodes during the encountering. Such information is used to build the social map on each node to reflect its understanding of the surrounding social structure. In the social map, the delivery ability between two linked nodes is evaluated by considering both meeting frequency and social closeness. As a result, each node only needs to check its own social map to decide whether to forward a packet to the newly met node, leading to efficient packet routing with a low cost. Extensive real-trace driven experiments show the effectiveness and efficiency of SMART in comparison with previous algorithms.

We further propose a method to realize efficient packet routing among landmarks, denoted by DTN-FLOW, in Chapter 6. We first split the entire MON area into sub-areas, each of which is represented by one landmark selected from frequently visited places in the network. Then, we measure the transmission capability between each pair of landmarks as the frequency of node transition between them. This is because packets can only be carried by nodes to move from one landmark to another in MONs. After this, we build a routing table on each landmark to indicate the next-hop landmark for each destination landmark with the smallest expected delay using the distance vector method. DTN-FLOW also predicts node transition using the order-k Markov predictor to forward packet from one landmark to a neighbor landmark. Finally, packets are routed one landmark by one

landmark to gradually reach their destination landmarks. Extensive analysis, experiments, and real deployment on our campus demonstrate the effectiveness of DTN-FLOW.

The future work will be in two directions. Firstly, the proposed methods in this dissertation can be enhanced by considering more realistic scenarios or including more useful information. For example, more realistic environment with dynamic file generation (i.e., file addition and deletion) and node request pattern can be considered to improve the adaptability of the theoretical analysis results and the proposed PCS method in Chapter 3. In SPOON, dynamically adjusting the thresholds for interest abstraction, community construction, and file searching can help find more suitable carriers for file requests and improve the file sharing efficiency. Other social factors, e.g., affiliation and interests, can also be considered to reveal more accurate social structures and mobility modeling in SMART and DTN-FLOW for more efficient packet routing.

Secondly, security and privacy issues are also critical for MON applications/services. While mobile devices are mainly held by people in MONs, the information exchanged among nodes in MON file searching or packet routing algorithms, e.g., real identifies, routing utilities, and selected packet forwarders, actually reflect people's personal privacy to a certain extent. For example, it can show who the device holder is, whom a person meets frequently, where a person visits for the most, and how tightly a person connects to others in a community. Such sensitive personal information can be exploited by adversaries to launch attacks in MONs. On the other side, while mobile services are usually deployed in open environments, nodes may not necessarily follow designed schemes or even launch certain attacks, e.g., attract and drop packets, to break the system. Therefore, it is essential to ensure privacy protection and security and meanwhile enabling efficient file searching or packet routing to encourage nodes to participate in MONs.

Bibliography

[1] http://web.informatik.uni-bonn.de/IV/BoMoNet/BonnMotion.htm.

[2] Dijkstra complexity. http://mathworld.wolfram.com/DijkstrasAlgori thm.html.

[3] GENI project. http://www.geni.net/.

[4] Orbit. http://www.orbit-lab.org/.

[5] The Network Simulator ns-2. http://www.isi.edu/nsnam/ns/.

[6] http://blogs.strategyanalytics.com/WDS/post/2012/10/17/Worldwide-Smartphone-Population-Tops-1-Billion-in-Q3-2012.aspx. Technical report, Strategy Analytics, 2012.

[7] George Aggelou. *Mobile Ad Hoc Networks: From Wireless LANs to 4G Networks.* McGraw-Hill Professional, 1 edition, 2004.

[8] Aruna Balasubramanian, Brian Levine, and Arun Venkataramani. Enhancing Interactive Web Applications in Hybrid Networks. In *Proc. of MOBICOM*, 2008.

[9] Aruna Balasubramanian, Brian Neil Levine, and Arun Venkataramani. DTN Routing as a Resource Allocation Problem. In *Proc. of SIGCOMM*, 2007.

[10] Chiara Boldrini, Marco Conti, Jacopo Jacopini, and Andrea Passarella. HiBOp: A History Based Routing Protocol for Opportunistic Networks. In *Proc. of WoWMoM*, 2007.

[11] Chiara Boldrini, Marco Conti, and Andrea Passarella. ContentPlace: Social-aware Data Dissemination in Opportunistic Networks. In *Proc. of MSWIM*, 2008.

[12] P. Bonacich. Factoring and Weighting Approaches to Status Scores and Clique Identification. *Journal of Mathematical Sociology*, 2(1):113–120, 1972.

[13] J. Broch, D. A. Maltz, D. B. Johnson, Y. Hu, and J. G. Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad hoc Network Routing Protocols. In *Proc. of MOBICOM*, 1998.

[14] John Burgess, Brian Gallagher, David Jensen, and Brian Neil Levine. MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networks. In *Proc. of INFOCOM*, 2006.

[15] Han Cai and Do Young Eun. Crossing over The Bounded Domain: from Exponential to Power-law Inter-meeting Time in MANET. In *Proc. of MOBICOM*, 2007.

[16] V. Carchiolo, M. Malgeri, G. Mangioni, and V. Nicosia. An Adaptive Overlay Network Inspired By Social Behavior. *Journal of Parallel and Distributed Computing*, 70:282–295, 2010.

[17] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott. Impact of Human Mobility on the Design of Opportunistic Forwarding Algorithms. In *Proc. of INFOCOM*, 2006.

[18] Augustin Chaintreau, Pan Hui, Jon Crowcroft, Christophe Diot, Richard Gass, and James Scott. Impact of human mobility on opportunistic forwarding algorithms. *IEEE Trans. Mob. Comput.*, 6(6):606–620, 2007.

[19] Kang Chen and Haiying Shen. Global Optimization of File Availability through Replication for Efficient File Sharing in MANETs. In *Proc. of ICNP*, 2011.

[20] Kang Chen and Haiying Shen. SMART: Lightweight Distributed Social Map based Routing in Delay Tolerant Networks. In *Proc. of ICNP*, 2012.

[21] Kang Chen and Haiying Shen. SMART: Utilizing Distributed Social Map for Lightweight Routing in Delay Tolerant Networks. *IEEE/ACM Trans. Netw., accpected*, 2013.

[22] Kang Chen and Haiying Shen. DTN-FLOW: Inter-Landmark Data Flow for High-Throughput Routing in DTNs. In *Proc. of IPDPS*, 2013.

[23] Kang Chen and Haiying Shen. DTN-FLOW: Inter-Landmark Data Flow for High-Throughput Routing in DTNs. *IEEE/ACM Trans. Netw., accpected*, 2013.

[24] Kang Chen and Haiying Shen. Maximizing P2P File Access Availability in Mobile Ad Hoc Networks through Replication for Efficient File Sharing. *IEEE Transactions on Computers, accpected*, 2014.

[25] Kang Chen, Haiying Shen, and Haibo Zhang. Leveraging Social Networks for P2P Content-Based File Sharing in Mobile Ad Hoc Networks. In *Proc. of MASS*, 2011.

[26] Kang Chen, Haiying Shen, and Haibo Zhang. Leveraging Social Networks for P2P Content-based File Sharing in Disconnected MANETs. *IEEE Trans. Mob. Comput.*, 13(2):235–249, 2014.

[27] X. Chen. Data Replication Approaches for Ad hoc Wireless Networks Satisfying Time Constraints. *IJPEDS*, 22(3):149–161, 2007.

[28] S. Chessa and P. Maestrini. Dependable and Secure Data Storage and Retrieval in Mobile Wireless Networks. In *Proc. of DSN*, 2003.

[29] Paolo Costa, Cecilia Mascolo, Mirco Musolesi, and Gian Pietro Picco. Socially-aware Routing for Publish-subscribe in Delay-tolerant Mobile Ad hoc Networks. *IEEE Journal on Selected Areas in Communications*, 26(5):748–760, 2008.

[30] Elizabeth M. Daly and Mads Haahr. Social Network Analysis for Routing in Disconnected Delay-tolerant MANETs. In *Proc. of MobiHoc*, 2007.

[31] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[32] W. P. Dotson and J. O. Goblen. A New Anylisis Technique for Probilisitic Graphs. *IEEE Trans. on Circults and Systems*, 26(10):855–865, 1979.

[33] H. Duong and I. Demeure. Proactive Data Replication Semantic Information within Mobility Groups in MANET. In *Proc. of Mobilware*, 2009.

[34] N. Eagle, A. Pentland, and D. Lazer. Inferring Social Network Structure using Mobile Phone Data. *Proceedings of the National Academy of Sciences*, 106(36):15274–15278, 2009.

[35] Kevin Fall. A Delay-Tolerant Network Architecture for Challenged Internets. In *Proc. of SIGCOMM*, 2003.

[36] Jialu Fan, Jiming Chen, Yuan Du, Wei Gao, Jie Wu, and Youxian Sun. Geocommunity-based Broadcasting for Data Dissemination in Mobile Social Networks. *IEEE Trans. Parallel Distrib. Syst.*, 24(4):734–743, 2013.

[37] Andrew Fast, David Jensen, and Brian Neil Levine. Creating Social Networks to Improve Peer-to-Peer Networking. In *Proc. of KDD*, 2005.

[38] Wei Gao and Guohong Cao. User-centric Data Dissemination in Disruption Tolerant Networks. In *Proc. of INFOCOM*, 2011.

[39] Wei Gao, Guohong Cao, Arun Iyengar, and Mudhakar Srivatsa. Supporting Cooperative Caching in Disruption Tolerant Networks. In *Proc. of ICDCS*, 2011.

[40] Wei Gao, Guohong Cao, Mudhakar Srivatsa, and Arun Iyengar. Distributed Maintenance of Cache Freshness in Opportunistic Mobile Networks. In *Proc. of ICDCS*, 2012.

[41] V. Gianuzzi. Data Replication Effectiveness in Mobile Ad-hoc Networks. In *Proc. of PE-WASUN*, 2004.

[42] R. S. Gray, D. Kotz, C. Newport, N. Dubrovsky, A. Fiske, J. Liu, C. Masone, S. McGrath, and Y. Yuan. CRAWDAD data set dartmouth/outdoor (v. 2006-11-06). http://crawdad.cs.dartmouth.edu/dartmouth/outdoor.

[43] Robin Groenevelt, Philippe Nain, and Ger Koole. The Message Delay in Mobile Ad hoc Networks. *Perform. Eval.*, 62(1-4):210–228, 2005.

[44] T. Hara. Effective Replica Allocation in Ad hoc Networks for Improving Data Accessibility. In *Proc. of INFOCOM*, 2001.

[45] T. Hara and S. K. Madria. Data Replication for Improving Data Accessibility in Ad Hoc Networks. *IEEE Trans. Mob. Comput.*, 5(11):1515–1532, 2006.

[46] D. W. Anna Hayes. Peer-to-Peer Information Sharing in a Mobile Ad hoc Environment. In *Proc. of WMCSA*, 2004.

[47] C. Hedrick. RFC1058 - Routing Information Protocol, 1988.

[48] Tristan Henderson, David Kotz, and Ilya Abyzov. The Changing Usage of a Mature Campus-wide Wireless Network. In *Proc. of MOBICOM*, 2004.

[49] C. Hoh and R. Hwang. P2P File Sharing System over MANET based on Swarm Intelligence: A Cross-Layer Design. In *Proc of WCNC*, pages 2674–2679, 2007.

[50] Weijen Hsu, Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Ahmed Helmy. Modeling Time-variant User Mobility in Wireless Mobile Networks. In *Proc. of INFOCOM*, 2007.

[51] Ying Huang, Yan Gao, Klara Nahrstedt, and Wenbo He. Optimizing File Retrieval in Delay-Tolerant Content Distribution Community. In *Proc. of ICDCS*, 2009.

[52] Pan Hui, Augustin Chaintreau, James Scott, Richard Gass, Jon Crowcroft, and Christophe Diot. Pocket Switched Networks and Human Mobility in Conference Environments. In *Proc. of WDTN*, 2005.

[53] Pan Hui, Jon Crowcroft, and Eiko Yoneki. Bubble Rap: Social-based Forwarding in Delay Tolerant Networks. In *Proc. of MobiHoc*, 2008.

[54] A. Iamnitchi, M. Ripeanu, E. Santos-Neto, and I. Foster. The Small World of File Sharing. *IEEE Trans. Parallel Distrib. Syst.*, 22(7):1120–1134, 2011.

[55] Adriana Iamnitchi, Matei Ripeanu, and Ian T. Foster. Small-World File-Sharing Communities. In *Proc. of INFOCOM*, 2004.

[56] Sushant Jain, Kevin R. Fall, and Rabin K. Patra. Routing in a Delay Tolerant Network. In *Proc. of SIGCOMM*, 2004.

[57] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *Proc. of ASPLOS-X*, 2002.

[58] J. Kangasharju, K. W. Ross, and D. A. Turner. Optimizing File Availability in Peer-to-Peer Content Distribution. In *Proc. of INFOCOM*, 2007.

[59] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* Wiley, New York, USA, 1990.

[60] L. Kleinrock. *Queueing Systems, Volume II: Coputer Applications.* John Wiley & Sons, 1976.

[61] A. Klemm, C. Lindemann, and O. Waldhorst. A Special-Purpose Peer-to-Peer File Sharing System for Mobile Ad Hoc Networks. In *Proc. of VTC*, 2003.

[62] Jani Kurhinen and Jukka Janatuinen. Geographical Routing for Delay Tolerant Encounter Networks. In *Proc. of ISCC*, 2007.

[63] Kevin C. Lee, Michael Le, Jerome Härri, and Mario Gerla. LOUVRE: Landmark Overlays for Urban Vehicular Routing Environments. In *Proc. of VTC Fall*, 2008.

[64] Kyunghan Lee, Yung Yi, Jaeseong Jeong, Hyungsuk Won, Injong Rhee, and Song Chong. Max-Contribution: On Optimal Resource Allocation in Delay Tolerant Networks. In *Proc. of INFOCOM*, 2010.

[65] Jeremie Leguay, Timur Friedman, and Vania Conan. DTN Routing in a Mobility Pattern Space. In *Proc. of WDTN*, 2005.

[66] Vincent Lenders, Martin May, Gunnar Karlsson, and Clemens Wacha. Wireless Ad hoc Podcasting. *Mobile Computing and Communications Review*, 12(1):65–67, 2008.

[67] Ilias Leontiadis and Cecilia Mascolo. GeOpps: Geographical Opportunistic Routing for Vehicular Networks. In *Proc. of WOWMOM*, 2007.

[68] Feng Li and Jie Wu. MOPS: Providing Content-Based Service in Disruption-Tolerant Networks. In *Proc. of ICDCS*, pages 526–533, 2009.

[69] Miao Lin, Wen-Jing Hsu, and Zhuo Qi Lee. Predictability of Individuals Mobility with High-Resolution Positioning Data. In *Proc. of UbiComp*, 2012.

[70] P. M. Lin, B. J. Leon, and T. C. Huang. A New Algorithm for Symbolic System Reliability Analysis. *IEEE Trans. Reliability*, 25(1):2–15, 1976.

[71] C. Lindemann and O. P. Waldhort. A Distributed Search Service for Peer-to-Peer File Sharing in Mobile Applications. In *Proc. of P2P*, 2002.

[72] Anders Lindgren, Avri Doria, and Olov Schelén. Probabilistic Routing in Intermittently Connected Networks. *Mobile Computing and Communications Review*, (3):19–20, 2003.

[73] J Link, Daniel Schmitz, and Klaus Wehrle. GeoDTN: Geographic Routing in Disruption Tolerant Networks. In *Proc. of GLOBECOM*, 2011.

[74] M. Lu and J. Wu. Opportunistic Routing Algebra and Its Application. In *Proc. of INFOCOM*, 2009.

[75] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a Feather: Homophily in Social Networks. *Annual Review of Sociology*, 27(1):415–444, 2001.

[76] S. Moussaoui, M. Guerroumi, and N. Badache. Data Replication in Mobile Ad hoc Networks. In *Proc. of MSN*, 2006.

[77] M. Musolesi and C. Mascolo. Designing Mobility Models based on Social Network Theory. *Mobile Computing and Communications Review*, 11(3):59–70, 2007.

[78] Mirco Musolesi and Cecilia Mascolo. CAR: Context-Aware Adaptive Routing for Delay-Tolerant Mobile Networks. *IEEE Trans. Mob. Comput.*, 8(2):246–260, 2009.

[79] A Nelson, J. R. Batts, Jr, and Robert L. Beadles. A Computer Program for Approximating System Reliability. *IEEE Trans. Reliability*, 19(2):61–65, 1970.

[80] M. Papadopouli and H. Schulzrinne. A Performance Analysis of 7DS: a Peer-to-Peer Data Dissemination and Prefetching Tool for Mobile Users. *Advances in Wired and Wireless Communications, IEEE Sarnoff Symposium Digest*, 2001.

[81] Joshua Reich and Augustin Chaintreau. The Age of Impatience: Optimal Replication Schemes for Opportunistic Networks. In *Proc. of CoNEXT*, 2009.

[82] T. Repantis and V. Kalogeraki. Data Dissemination in Mobile Peer-to-Peer Networks. In *Proc. of MDM*, 2005.

[83] Sebastian Schnettler. A Structured Overview of 50 Years of Small-world Research. *Social Networks*, 31:165–178, 2009.

[84] Hinrich Schüze and Craig Silverstein. Projections for Efficient Document Clustering. In *Proc. of SIGIR*, 1997.

[85] Gaurav Sharma, Ravi Mazumdar, and Ness B. Shroff. Delay and Capacity Trade-Offs in Mobile Ad hoc Networks: A Global Perspective. In *Proc. of INFOCOM*, 2006.

[86] Libo Song, David Kotz, Ravi Jain, and Xiaoning He. Evaluating Location Predictors with Extensive Wi-Fi Mobility Data. In *Proc. of INFOCOM*, 2004.

[87] T. Spyropoulos, K. Psounis, and C. Raghavendra. Efficient Routing in Intermittently Connected Mobile Networks: The Single-copy Case. *IEEE/ACM Trans. Netw.*, 16(1):63–76, 2008.

[88] J. B. Tchakarov and N. H. Vaidya. Efficient Content Location in Wireless Ad hoc Networks. In *Proc. of MDM*, 2004.

[89] Amin Vahdat and David Becker. Epidemic Routing for Partially-Connected Ad hoc Networks. Technical report, Duke University, 2000.

[90] Jie Wu, Mingjun Xiao, and Liusheng Huang. Homing Spread: Community Home-based Multi-copy Routing in Mobile Social Network. In *Proc. of INFOCOM*, 2013.

[91] L. Yin and G. Cao. Supporting Cooperative Caching in Ad hoc Networks. *IEEE Trans. Mob. Comput.*, 5(1):77–89, 2006.

[92] Eiko Yoneki, Pan Hui, ShuYan Chan, and Jon Crowcroft. A Socio-aware Overlay for Publish/-Subscribe Communication in Delay Tolerant Networks. In *Proc. of MSWiM*, pages 225–234, 2007.

[93] Quan Yuan, Ionut Cardei, and Jie Wu. Predict and Relay: An Efficient Routing in Disruption-tolerant Networks. In *Proc. of MobiHoc*, 2009.

[94] J. Zheng, J. Su, K. Yang, and Y. Wang. Stable Neighbor Based Adaptive Replica Allocation in Mobile Ad Hoc Networks. In *Proc. of ICCS*, 2004.