Noname manuscript No.
(will be inserted by the editor)

MobiStore: A System for Efficient Mobile P2P Data Sharing

Mohammad A Khan · Laurent Yeh · Karine Zeitouni · Cristian Borcea

Received: date / Accepted: date

Abstract MobiStore is a P2P data store for decentralized mobile computing, designed to achieve high availability and load balance. As P2P platforms, mobile devices connected to the Internet through WiFi or cellular networks are different from wired devices in two main aspects: (1) higher churn due to mobility, weak wireless signals, or battery constraints, and (2) significant variability in bandwidth and latency based on the point of attachment. These problems affect the stored content availability and skew the content serving load over the peers. MobiStore structures the mobile P2P network into clusters of redundant peers. The topology uses both algorithmically-defined and random edges among the peers of different clusters. The routing information is updated using a gossip-based protocol. Thus, MobiStore achieves, with high probability, $O(1)$ lookup operations despite high churn and link variability. Inside the clusters, all peers replicate the content, which improves the content availability. Furthermore, based on the current load, MobiStore dynamically changes the number of peers inside the clusters and routes content request to randomly selected peers. These two dynamic techniques along with using consistent hashing to map content to peers, balance the load over the peers. While some of these techniques are well known, the main contribution is on the novel ways of applying them to design and implement an efficient mobile P2P data store. Simulation results show MobiStore achieves an availability, i.e., lookup success rate, between 12%-48% higher than two baseline systems built over the MR-Chord and Chord P2P protocol; and reduces the latency up to 9 times. Finally, the results show MobiStore adapts to churn and workload

Mohammad A Khan, Cristian Borcea
Department of Computer Science, New Jersey Institute of Technology, Newark, NJ, USA.
E-mail: mak43,borcea@njit.edu

Laurent Yeh, Karine Zeitouni
Department of Computer Science, University of Versailles Saint-Quentin-en-Yvelines, France.
E-mail: laurent.yeh,karine.zeitouni@uvsq.fr

to evenly distribute the requests across clusters and peers better than both baseline solutions.

Keywords Mobile P2P, P2P data store, availability, load balance

1 Introduction

Smart phones and tablets are quickly becoming the main computing devices in our daily life [24]. The amount of mobile user-generated content and mobile sensing data will become very large in the near future. For example, estimates predict over 5 petabytes of data generated every day by mobile phone subscribers around the world [10]. These data will enable novel mobile applications that provide new and rich user experiences. For example, users can run applications that tell them the restaurants visited by their friends in a certain city, search for a lost child face among the photos taken by people nearby, or find out about traffic congestion along their driving routes.

Currently, some of these applications could be implemented based on the client-server model with potential help from the cloud. Service providers could offer such applications for free in exchange for the ability to collect user data and analyze user behavior, which they can monetize by selling it to advertisers, etc. Instead of letting service providers collect user data, many users would like to own and control their mobile data [34]. They may, however, be willing to share the data (or share the results of computations over this data) with communities defined by friendship, similar interests, and/or geography. This scenario lends itself naturally to mobile peer-to-peer (P2P) computing which enables direct and scalable collaboration among mobile users. Unlike mobile ad hoc networks which assume no Internet connectivity, mobile P2P computing assumes mobile devices are connected to the Internet using cellular or WiFi networks.

Despite all the advances in traditional P2P networks, there is still a dearth of efficient solutions for mobile P2P networks. For example, we are not aware of any practical implementation that works well on smart phones (i.e., has good availability, scalability, and latency). Partially, this could be due to the lack of killer apps for mobile P2P. However, as we argue in this article, this situation is likely to change dramatically in the near future. The technical reason for the absence of efficient mobile P2P deployments is that mobile devices are different from wired devices as P2P platforms in two main aspects: (1) higher churn due to short wireless sessions, which are the results of mobility (e.g., connecting to different wireless networks while on the move, change of WiFi access points which changes the IP address) or user choice (e.g., turning the device off to save battery power, turning the cellular data connection off to avoid usage above the contract limits, etc.), and (2) link quality variability as bandwidth and latency change drastically based on current point of attachment. The question, then, is: can existing P2P solutions for wired devices work well in a mobile environment?

Structured P2P solutions using DHTs maintain rigorous geometric topologies for routing resiliency. But these topologies require constant maintenance which substantially increases the overhead in the presence of high churn. Furthermore, churn can make routing tables inconsistent, and this increases lookup latency and failure rates and can even partition the network [29], [32], [3]. Structured P2P solutions for coping with churn use link delay estimations [29] and proximity, but they cannot be employed for mobile P2P networks as the values of these parameters change over time for mobile peers. In addition, unstructured P2P solutions do not work in this type of environment due to their low efficiency.

Existing mobile P2P solutions, on the other hand, mainly improve the routing information management process of the structured P2P solutions using real-time updating of finger tables [35], [12] or maintaining a secondary set of finger table entries [14]. However, maintaining routing information which requires multiple fail-prone peers to work in unison increases the chance of failure. Therefore, existing solutions do not work well for high churn scenarios. Thus, in summary, current P2P solutions are inadequate for mobile P2P computing.

This article presents MobiStore, a mobile P2P data store that uses redundant peers and clustering to compensate for the side effects of P2P churn and link level variability, particularly: low availability and skewed request load distribution. MobiStore structures the mobile P2P network into clusters of redundant peers, called Virtual Peers (VP). The VPs maintain a network structure consisting of both algorithmically-defined and random edges to one another. The inter-VP routing information is updated using gossiping. Inside VPs, members are fully connected as the cluster sizes are relatively small. MobiStore achieves $O(1)$ lookup operations with high probability, and its hierarchical structure makes the topology robust to churn.

The mobile peers in each Virtual Peer (VP) replicate the keys and data assigned to their VP. Thus, any VP member can answer queries for its VP. A lazy-update protocol is used to maintain weak consistency of the stored content among VP members. As multiple members can answer the same queries, the effect of individual churn is minimized. To simplify routing, the VPs have static IDs, managed by MobiStore seamlessly in a decentralized manner. Similarly, mobile peers are assigned static IDs at the time they first join the network, thus decoupling peer naming from IP addresses. In this way, rejoining the network incurs reduced overhead because under normal conditions mobile peers re-join the same VP they have previously left. To minimize the required bandwidth for topology management update propagation, MobiStore uses a hierarchical update process. Load balance is achieved through: (1) storing data using consistent hashing over VPs, and (2) load adaptive VP management which spreads the lookup requests randomly over the members of a VP and varies the number of peers in the VP depending on the content popularity.

We evaluated our system over PeerSim [22] using extensive simulations with 6,500 peers, which connect to the Internet over WiFi and cellular networks. To test MobiStore's improvements over existing P2P solutions, we built

baseline data stores having similar content handling techniques as MobiStore but working over MR-Chord [35] [1] and Chord-based P2P network [33]. The results show MobiStore achieves a lookup success rate (which measures availability) between 12% and 40% higher than the solution based on MR-Chord, and between 14% and 48% higher than the solution based on Chord. MobiStore also reduces the latency up to 9 times compared to MR-Chord or Chord based solutions. MobiStore achieves these benefits by increasing the management traffic overhead by only 1-3KB per minute over the solutions based on MR-Chord and Chord. We found that MobiStore can make new content visible to everyone very fast (only limited by the content size and network bandwidth). Finally, MobiStore adapts to churn and workload to evenly distribute the requests across peers better than both MR-Chord and Chord solutions.

The rest of this article is organized as follows. Section 2 presents the overview of MobiStore, and the related work is discussed in Section 3. Section 4 details the design of MobiStore's core elements. Section 5 presents the simulation results and analysis. The article concludes in Section 6.

2 Overview of MobiStore

Figure 1 shows the high level structure of MobiStore, in which the peers are divided into virtual peers (VPs) connected using a robust topology. A VP is a clique of peers (i.e., fully connected peers) which replicate the stored content of the VP and, thus, provide availability and load balance. The topology is formed by two types of inter-VP edges: (1) edges to maintain a Chord-like ring [33] for guaranteed topology update performance ($O(\log N)$), and (2) random edges to improve the topology update performance further. Unlike traditional P2P protocols, these edges are not used to route store/lookup requests; they are used just for topology maintenance. Each peer in a VP maintains this topology information in the VP finger table.

VPs periodically exchange their finger tables (i.e., done by the peers inside VPs), merge the information received from others, and form their aggregate routing tables. These tables contain routing information to all the other VPs, thus making routing in MobiStore work in $O(1)$ with high probability. In rare situations, due to delays in topology update information synchronization, the routing will work in $O(\log N)$ using the ring. In this process, the peers minimize the routing latency and increase topology stability at the cost of slightly higher computation, communication and storage resources than existing P2P protocols.

Figure 2 shows the data structures used to maintain the topology and routing information: the *local routing table* is used for local routing inside VP, which includes the current type of Internet connection (WiFi or cellular); the *VP finger table* is the topology information table formed over VPs using the Chord-like ring and the randomized edges; and the *aggregate routing table* is

[1] MRChord is a very recent version of Chord that targets mobile peers.

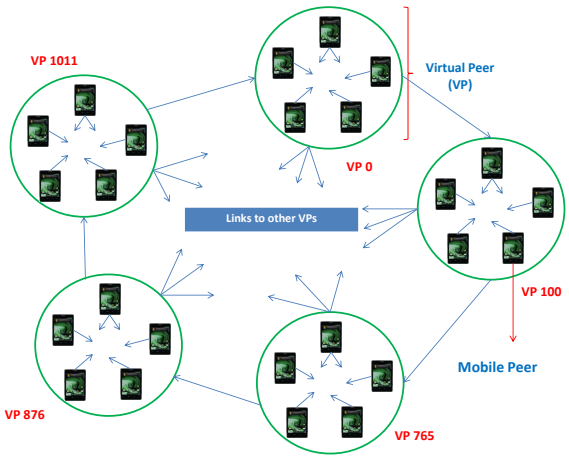

Fig. 1 Structural overview of MobiStore

Aggregate routing table (for VP0)

VP ID	IP List	Retry Count
1	65.63.12.23, 97.98.23.23, … … …	12
2	34.23.89.11, … … …	2
4	…. … …	1
7	… … …	…
8	… … …	…
… …	… … …	…

Global VP statistics table

VP ID	Bandwidth Usage (KB/per minute)	System UP probability	Peer Count
0	2.0	0.75	10
1	1.667	0.56	15
2	1.0	0.90	15
4	4.234	0.85	35
5	3.3	0.40	26
6	4.0	0.33	6
…			

VP finger table (member of VP X)

VP ID	IP List
X+1	34.56.78.123, 65.63.12.23, … … …
X+2	34.56.19.11, … … …
… …	…. … …
Random1	128.21.22.98,…
Random2	66.32.12.122, …
… …	… … …

Local routing table (members of my VP)

Peer ID	Peer IP	Current Interface
GHTRE	34.56.78.124	WiFI
LKUYT	123.56.78.123	Cellular
JKTRW	45.76.23.12	Cellular
DF231	90.34.12.11	WiFi

Fig. 2 Routing and management information maintained by the peers

formed by merging the VP finger tables received from other VPs. The retry count column in this table is used to asynchronously update the routing entries (without waiting for the update intervals). Every time a request sent to a VP fails, the associated retry count is incremented. If the retry count reaches a predefined threshold value, a request for the current *local routing table* is sent to all the members found from that entry. The existing members will reply with the correct routing table while the non-existent members (who were responsible for the errors) will not answer, thus correcting the entry. Then, the entry is updated with the new IP addresses.

The *global VP statistics table* stores dynamic system statistics to maintain availability and load balance. The statistics include bandwidth usage over the VPs, average up-time of the VP members, and VP member counts. For

example, when a new peer joins, it should be added to a VP with a low number of peers or to a VP with low average up-time. The bandwidth column in this table is used for balancing the bandwidth usage load among the peers.

Inside VPs, peers use a peer-to-peer rumor propagation process to synchronize these tables before sending updates to other VPs. The rumor propagation reduces the amount of bandwidth used for synchronization and provides reliability to churn. Although VPs contain many peers, only a few peers from each VP send updates to other VPs (VP-VP communication) to maintain scalability and save bandwidth.

Having multiple mobile peers assigned to each VP allows MobiStore to provide high availability through replication. Key-values are mapped over the VP identifier space. All mobile peer members of a VP store the same key-value pairs. For example, in Figure 1, if key K_1 maps to VP 100, all the members of VP 100 will store the values of K_1. Thus, any VP member can answer queries for its VP.

Load balancing is achieved by spreading the requests uniformly over the peers of a VP. If the load on each peer of a VP becomes too high, MobiStore dynamically adds more peers in this VP; these peers are taken from lightly-loaded VPs.

3 Related work

There are numerous well-known P2P protocols to efficiently manage key-value pairs, such as Chord [33], Pastry [30], CAN [27], Tapestry [36], etc. All these protocols manage key-value pairs efficiently and have robust routing mechanisms, but they have problems with high churn. It has been shown that these protocols increase the routing latency or even partition the network under churn [29,32]. There are a few solutions [29] for this problem, but they are dependent on the link delay estimation. For mobile peers, the link delay is difficult to estimate because it changes depending on the point of attachment. In addition, these solutions depend on fixed IP addresses to map content to peers. For mobile peers, the IP addresses change quite frequently. Therefore, these existing solutions are inadequate to serve for mobile P2P computing.

Mobile robust Chord (MR-Chord)[35], MChord [21], and opChord [14] are recent works designed to solve the mobility-related issues and design P2P networks which improve the routing success and decrease the lookup latency. MR-Chord[35] improves Chord's finger maintenance process by using failure statistics to pro-actively update finger entries in real-time. MChord[21], designed for mobile ad hoc network environments, employs several techniques such as vigorous finger table updates using information from every possible snooped message, exchanging entire finger tables, etc. opChord[14] maintains secondary sets of finger tables to point at more peers covering additional regions in the ring space, which is used for improving routing efficiency. However, all these works still maintain multi-hop routing information. MobiStore, on the other hand, maintains 1-hop routing information, which has the potential to

improve availability by minimizing the number of failures during request forwarding. MobiStore reduces the impact of the inconsistency due to high churn by using redundancy and randomization, while correcting the inconsistency periodically.

C-Chord [37] and Chordella [12] explore different aspects of mobile P2P systems. The goal of C-Chord[37] is to localize P2P traffic among the users of the same base station to reduce network traffic. This system works only for cellular users. However, most users employ WiFi most of the time, especially when transferring large amounts of data. Therefore, this solution's scope is not general enough for a mobile P2P platform. Chordella[12], on the other hand, uses PCs (super peers) to maintain the DHT ring, and mobile devices connect to these super peers. Unlike Chordella, MobiStore presents a mobile-only solution that does not require help from wired systems, which introduce an extra-complexity layer that may preclude simple deployment. In addition, unlike these systems, MobiStore improves the content availability and the load balance of the system.

Hierarchical P2P systems generally assign special roles to some peers. Authors in [6] proposed a load balance scheme for P2P file search using a three-level hierarchical P2P network. The lowest layer stores content, and the top two layers store information about their bottom layers. The indirections in the top layers are exploited to balance the load. However this type of architecture requires special maintenance for top level peers. Hivory [23] uses a multilevel hierarchy designed as a tree of Voronoi planes to store multi-attribute information. Synapse [20] proposes protocols to interconnect and provide collaboration among heterogeneous overlay networks. MobiStore exploits hierarchy to minimize the topology maintenance traffic, improve resiliency in routing, and better manage the redundancy. However, it does not assign special roles to certain peers, and thus it is more resilient to churn.

Similar to MobiStore, the works in [13,31] assume that mobiles communicate with each other over the Internet and form P2P networks for data sharing and distributed computations. However, they have significant problems in larger scale networks: latency (due to cellular communication), availability (due to change in IP addresses and resource limitations), and load balancing (generally not considered in the design). MobiStore solves these issues through its hierarchical and highly available network structure, which works well in the presence of short wireless sessions and resource limitations.

In the past decade, there have been several attempts to leverage DHTs in mobile ad hoc networks and sensor networks [28,2,4,17]. Similarly, several projects have proposed data management and services over opportunistic mobile networks [7,19,5,25]. Due to the high volatility nature of these networks, the proposed solutions cannot hope to acquire a global view of the network. Thus, their algorithms must be localized: they are good for scalability, but they provide poor results in terms of availability, latency, and load balancing. MobiStore solves these issues through its Internet-based mobile P2P solution, which allows peers to acquire a weakly consistent global view of the network.

The principle of using multiple physical peers and keep them as a group (similar to our VPs) has been discussed in mDHT [18] and Kelips [11]. mDHT uses all the peers in a subnet as one super peer in the DHT ring. This facilities the use of the default Ethernet link-level multicast to optimize the network traffic. This idea is not suitable for mobile peers communicating over the Internet. Kelips's main goal, on the other hand, is to make routing fast, i.e., O(1). The peers in Kelips store a large number of IP address-to-peer mappings as well as file-to-peer IP address mappings. While churn is not a issue for an entire group, it becomes a problem if a peer storing a file fails or disconnects abruptly. Unlike MobiStore, Kelips has no techniques to maintain content availability in the network.

A few solutions discuss the load balance problem for P2P networks. Chord [33] addresses load balance by assigning several logical peers for one physical peer. This provides finer-grain control over the mapped key-values, which can be exploited to balance the load. Further improvements of this idea using different methods of moving the logical peers have been discussed in [15,26]. These approaches have two problems: First, all the logical peers act as an actual peer and generate too much background traffic. Second, the effect of churn increases. Therefore, these solutions are not suitable for mobile peers.

Several P2P file systems such as Past [8] and Oceanstore [16] have been built on top of DHT-based P2P protocols [30,36]. These systems inherit the problems related to churn that are associated with their routing protocols. In addition, the overhead of building complete file systems can be an overkill for typical mobile applications which work well with key-value data stores. Such file systems bring not only extra-complexity, but also additional overhead.

4 Core Design Elements

This section discusses the core design elements of MobiStore: (1) how do peers join the network and what happens when they leave? (2) how to set the number of peers in a VP? (3) how to maintain the network topology and the routing tables at peers? (4) how to provide load balance? (5) how to store and retrieve content?

4.1 Peer Join/Leave

A peer needs to know at least one network member to join the system. A peer receiving a join request checks the total number of peers in its own VP and, in case the number is lower than a threshold value, the new peer is added to the same VP. Otherwise, the peer receiving the join request uses its global VP statistics table to find a suitable VP for the new peer. First, it attempts to assign the new peer to a VP which is suffering from low availability (found from the statistics update packets). If the availability is within limits, the new peer is assigned to the VP which is currently receiving the highest rate of

lookup requests. In this way, MobiStore is able to use the new peers to improve availability and load balance.

The new peer receives the ID of its VP and the list of other VP members. Then, it sends a join request to any of these members. Finally, it creates its fixed random ID that decouples its naming from IP addressing and sends the ID to the other members of the VP. The new peer receives a copy of the routing table as well as the statistics table from the other peers.

The VP which added the new peer splits into two VPs if the joining of the new peer exceeds the maximum allowed number of peers in the VP. One of the newly created VPs remains in the same position of the Chord-like ring, and another one randomly selects a position between the existing VP and its current successor. The two VPs and the current successor update their routing tables accordingly and send the information to everyone in the next global update interval. At the same time, the new VPs add a number of random links to other existing VPs.

The above mentioned steps work well for network formation. A new peer starts the network with just one VP and waits for new members. New peers are added to the same VP until the maximum threshold value is reached. After that, any new peer join request initiates a VP split and executes the steps mentioned in the previous paragraph. As more members join the network, these two VPs split again. The process continues over the lifetime of the system.

When a peer leaves the system gracefully, it sends a notification to the members of the VP. The members receiving the request remove its IP from their lists. It is assumed that a peer which changes its IP will inform the other members about the change. To collect system and network statistics, peers exchange updates including their IP addresses periodically. Therefore, if a peer leaves the system suddenly without informing the other peers, these peers will notice after a while from the statistics; alternatively, they notice when a request to that peer fails.

4.2 Number of Peers per VP

One of important question in our design is "how many peers should form a VP?". The answer depends on the designer's goals. It is possible to fix the peer count to satisfy certain availability and load balance criteria. In the following, we discuss two heuristics to determine the number of peers in a VP.

Satisfying the availability criterion: The number of peers per VP to maintain a certain availability of stored content can be found by using the following equations. Let, the probability of finding an individual peer up and running be up, and the probability of a successful retrieval at any time be P. Then

$$P = 1 - (1 - up)^m \tag{1}$$

where m is the number of peers in the VP. Thus:

$$m = \frac{\log(1-P)}{\log(1-up)} \tag{2}$$

Therefore, we can determine m, the number of peers in a VP, by fixing P to a value such as 0.99. The value for up can be found from system statistics.

Satisfying the load balance criterion: We fix the peer resource utilization (computation, network bandwidth, etc.) and determine the number of peers per VP as a function of the offered load in order to maintain roughly the same load for each peer. A VP can be modeled as an M/M/m queue. The number of peers in the VP is m. Then, the peer usage is:

$$Util = \frac{\lambda}{up * m * \mu} \tag{3}$$

Here, λ is the incoming request rate for a VP, μ is the average serving rate for a peer, and up is the probability of a peer being up. It is possible to determine m by fixing the other parameters. For example, to find a system where every peer roughly receives the same workload, we determine m by fixing peer utilization, $Util$, to a predefined value and finding λ, μ, and up from the dynamic system statistics. To determine m in a system where everyone roughly spends the same fraction of bandwidth, we can fix $Util$ as the fraction of the total data capacity limit, μ as the serving rate of the bandwidth, and the other parameters the same as before.

In our implementation, we use the first heuristics (satisfy the availability criterion) and then dynamically change the peer count per VP to keep the average bandwidth usage rate for answering the queries constant. In a different implementation, one could use the second heuristic to optimize for load balance.

4.3 Topology and Routing Table Maintenance

MobiStore maintains the topology, routing tables, and network statistics using hierarchical updates, which improve scalability and fault-tolerance. The data structures used in this process are shown in Figure 2. Updates are periodically exchanged and synchronized among the peers inside VPs. Then, VPs exchange the synchronized updates periodically as well. The update intervals for the peers inside VP (intra-VP updating) have shorter duration than update intervals among the VPs (inter-VP updating). Therefore, peers inside the same VP can synchronize the information among them before sending it to other VPs.

In the following, we describe how the updates are disseminated, how the individual tables (Figure 2) are updates, and how the update process is made robust to churn.

4.3.1 Intra-VP update dissemination

Intra-VP updates use a peer-to-peer gossip-based dissemination protocol. Every peer sends updates to randomly selected $\log M + c$ peers periodically (i.e., every local-VP update interval). M is the number of peers inside the VP and c is a constant. Peers always send most up-to-date tables.

4.3.2 Inter-VP update dissemination

Inter-VP updates are sent to $\log N + c$ other VPs periodically (i.e., every global update interval); $\log N$ are the Chord finger entries and c are the random edges. Let us emphasize that VP-to-VP communication means that a certain number of randomly selected peers from the sender VP (the number depends on the specific operation) communicate with a certain number of randomly selected peers from the destination VP.

Each peer sends an update with probability $\frac{p}{M}$, where p is a predefined constant and represents the number of peers from the VP that send updates at each interval, and M is the total number of peers in that VP. To maintain fault-tolerance and minimize the propagation latency, the peers send updates to q random members of each receiving VP. Selecting large values for p and q would increase the chance of successful propagation, but would use more bandwidth. Roughly, with this method, during each update interval, each VP sends $p \times q \times (\log N + c)$ packets, where N is the number of VPs and c is the number of random edges. The number of update packets is practical for current networks and devices: suppose a VP has 20 peers and $p = 4, q = 4, \log N = 15, c = 4$. Then, each VP sends 304 messages per update interval. This is roughly 15.2 messages per peer per VP, which is quite low considering the network size of 655,360 (20×2^{15}) peers. In addition, the size of update messages is always in the order of several KBs, and the global update intervals are on the order of minutes.

4.3.3 Updating the VP finger table

The *VP finger table* contains Chord-based finger entries for the VPs along with entries for some random edges. The finger entries are updated according to the original Chord protocol. The random edges are updated when failures are detected, and their number is predefined and does not grow with the network size. MobiStore uses this table for VP-to-VP communication and inter-VP synchronization.

4.3.4 Updating the aggregate routing table and the global VP statistics table

The base routing information from the individual VP finger tables, which are exchanged among the VPs periodically, is used to compose the *aggregated routing table*. Eventually, this table will contain entries for all the VPs, not only for those existing in the local VP finger tables. Thus, it will enable $O(1)$

routing. Let us note that the aggregated routing tables of different VPs may sometimes be inconsistent, but this issue is solved through periodic inter-VP synchronization. This synchronization is done over the VP finger table entries. Each VP sends both the aggregated routing table and VP finger table even though sending only the former is enough. This is done to improve fault-tolerance and minimize the update propagation latency. In case a VP misses some updates, other VPs are going to send those updates in the aggregated tables. Therefore, VPs need not wait until the next interval to receive the information.

The *global VP statistics table* stores dynamic system statistics for load balancing. Peers inside a VP exchange information about bandwidth usage and up time. After receiving this information from all the members of the VP, any peer can calculate the averages for the VP and update the local copy of the global VP statistics table. The global VP statistics tabled are synchronized in the same way as the aggregated routing tables.

The inter-VP updates are disseminated to every VP in the system in $o(\log N)$ global update intervals. MobiStore uses the VP finger table to propagate the updates. In a Chord-based system, the update propagation tree has a height of $\log N$. In addition to the finger entries, MobiStore uses the random edges which can reduce the number of intervals needed to send the updates to everyone. For example, the random edges could be pointing to the lower level of the propagation tree. Therefore, some peers will receive the updates in less than $\log N$ update intervals.

4.3.5 Updating the local routing table

The local routing table is used for local routing inside the VP, which is achieved in $O(1)$. The table is updated each time a peer changes its IP address, which results in an update message with the new IP to the rest of the members of the VP. Therefore, all members of the VP are always up-to-date. As the member count of a VP is low, the updates are not a significant burden. Nevertheless, a number of simple optimizations are done to improve the efficiency of this process: (1) wait for a couple of minutes to detect if the current IP is stable (when the peer is on the move), (2) send the new IP to a few peers who are using WiFi and they can broadcast it, thus reducing the effect of the cellular communication latency.

4.3.6 Robustness of the update process

Both intra and inter VP update processes send updates to $\log(total) + C$ peers, where C is a constant. Intra-VP updating uses gossiping, and inter-VP updating uses gossiping along with fixed edges updates (dictated by the finger table). This process has been shown to succeed in reaching everyone with high probability [9]. For example, $C = 4$ can achieve 98.18% chance of success (which is expected to be higher for MobiStore due to aggregated updates).

4.4 Load Balance

Load balance is achieved through three methods: (1) consistent hashing to store data in VPs, (2) randomization for each operation to spread the load evenly (exploiting redundancy) and to limit the effect of temporary failures, and (3) load adaptive VP management which varies the number of peers in the VP proportionally to the bandwidth needed to answer the queries. Let us note that MobiStore considers load balance over longer durations (e.g., do not consider flash crowds).

4.4.1 Uniformly spreading the requests

In MobiStore, each peer keeps multiple pointers to different members of each VP. Therefore, peers send lookup/store requests to a random peer in a VP for each request. Furthermore, each peer knows how much data it served and how much are served by other members of the VP (from the statistics table). In case of an imbalance, the affected peer forwards the request to a less used peer.

This method works if the number of peers in a VP is proportional to the incoming request rates. If not, MobiStore can change the number of peers in a VP, create a new VP, or merge VPs.

4.4.2 Changing the number of peers in VP

Algorithm 1, shows the logic used to change the number of peers inside the VPs. During peer join, some peers (with certain probability), are treated as loose peers. Only loose peers can move from one VP to another VP. They are always temporary members of the VPs. In this way, the system can maintain stability as most peers get permanent membership to VPs. The proportion of loose peers is a design trade off: a high number could result in an unstable system, while a low number could not balance the load well.

The algorithm shows the conditions of moving a peer: the peer must be a loose peer, must be the member of a VP which currently uses the least bandwidth to answer queries, and must have changed its VP a long time ago (to amortize the content movement cost). If these conditions are satisfied, the peer moves to the VP with the highest load, as measured by its average bandwidth utilization; this utilization is retrieved from the global VP statistics table. A VP has a limit on how many peers it can keep. Once a peer decides to move to another VP, it sends a peer leave message to all its current VP members, and then joins the new VP.

While one peer is moving, no other peer attempts to move. Peers exchange this information in the statistics update packets. Moved peers check for errors after a predefined time and can roll back if errors are detected.

ALGORITHM 1: Load adaptive member count for VP executed by peers

```
Data structures
  Some data structures are updated by statistics and routing modules
  my_vp_members: IP list of the members of this VP
  routing_table: aggregated routing table
  finger_table: base routing finger table
  my_stat_table: local peer statistics
  global_stat_table: statistics for all peers
  Constant Input parameters
  MAX-BANDWIDTH-LIMIT: limit after which load balance starts
  MAX-ALLOWED: maximum number of peers allowed in a VP
  VP-CHANGE-INT: time interval before a peer can change its VP
  DEF: probability a peer waits for another peer to start a new VP
  Other variables set in previous iterations or by other modules
  loose-peer ← amILoosePeer(my_stat_table)
  min_band ← getMinBandwidthUsingVP(global_stat_table)
  max_band ← getMaxBandwidthUsingVP(global_stat_table)
  max_vp_count ← memberCountMaxBandwidthVP(global_stat_table)
  new_vp ← idOfMaxBandwidthVP(global_stat_table)
  my_band ← myVPBandwidthUsage(my_stat_table)
  last_vp_change ← curTime() - lastTimeVPchange(my_stat_table)

update_variables(global_stat_table)
  if (loose-peer AND my_band == min_band AND last_vp_change > VP-CHANGE-INT)
  then
     if (max_band > MAX-BANDWIDTH-LIMIT) then
        if (max_vp_count < MAX_ALLOWED) then
           for p in my_vp_members do
              send_permanent_leave(p,my_id);
           end
           new_vp_members ← get_member_list(new_vp);
           for p in new_vp_members do
              send_load_bal_join(my_id,p);
              receive_routing_stat_update_from(p);
           end
        end
        prob ← uniform_real_prob(0,1);
        if (prob > DEF) then
           wait_for_another_moverequest_from_neighbors();
        end
        start_a_new_vp_between_max_and_neighbor();
     end
end
```

4.4.3 Terminating existing VPs

An existing VP is terminated if it had redistributed all its loose peers and it still is the VP with the lowest incoming request rate. In this situation, all the permanent VP members are distributed to other VPs. To begin the process, the VP marks itself for re-distribution and sends this information at the next global update interval to the other VPs (piggy-backed on the global statistics messages). After receiving acknowledgments from all the other VPs confirming that this marked VP was removed from the network, the peers from this VP

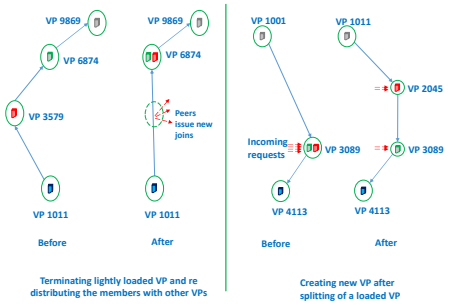

Fig. 3 Load Adaptation in MobiStore: terminating VPs and splitting VPs

send the stored content to the next clockwise VP and issue new join requests, as illustrated in the left part of Figure 3. By default, a peer submitting a join request is assigned to the VP with the maximum load. Due to inconsistent information, it is possible that more than one VP will attempt to remove itself from the network at the same time (i.e., the same global interval). In such a situation, the other VPs send negative acknowledgments and the marked VPs back-off randomly before attempting to remove themselves from the network again.

4.4.4 Creating new VPs

If a VP has already added loose peers and has reached the maximum number of peers a VP can have, and it still experiencing a request rate higher than a certain threshold, it splits itself in two VPs. This process is depicted in the right part of Figure 3. One of these VPs maintains the old VP ID, and the other takes an ID between the values of the old ID and one of its two direct neighbors on the virtual ring. The neighbor with the largest distance between the IDs is considered, and the new ID will be set to the half of this distance in the ID space. Each peer inside the VP that splits randomly decides on which of the two VPs it will be a member with probability 0.5. The new VP issues a join request as a virtual ring node (i.e., VP), and after the join is complete, it sends global updates to all the other VPs.

4.5 Content Storage and Retrieval

Consistent hashing together with the robust ring-like structure of the VPs ensure good load balance for data placement and scalability. Since recent mobile devices have large storage capacity, MobiStore assumes that peers in the network have enough storage for data sharing and does not deal with data eviction.

In MobiStore, keys and VP IDs are mapped over the same address space. When a mobile peer performs a key lookup, it uses the aggregate routing table to find the 1-hop routing entry to the destination VP. In the unlikely case that such an entry does not exist yet, MobiStore falls back to using the finger entries. This case can happen for the time between joining the network and building the aggregate routing table (which requires several global update intervals). An entry in the aggregated routing table contains several IP addresses. The peer randomly chooses one to send the request. If the requested peer does not have the content yet (due to delayed synchronization within the VP) or it has already used more bandwidth than other peers within the VP, it forwards the lookup request to another peer. In our implementation, a request can be forwarded at most 3 times.

Once new content is stored at a peer, this peer uses a gossip based protocol to send the new content to everyone in the VP. To minimize the cellular network bandwidth requirements, only peers using WiFi take part in the content update process. The peers currently on cellular network, start getting updates when they switch to WiFi.

5 Evaluation

We used PeerSim, a P2P Java based simulator [22] to evaluate MobiStore. Our experiments compare MobiStore with two baseline data-stores built over MR-Chord (Mobile Robust Chord) and Chord. PeerSim provides the Chord implementation. Since we could not find any publicly available implementation of MR-Chord, we implement it based on its description [35].

MR-Chord improves the finger management process of original Chord protocol. Each time a peer experiences a routing failure, it sends a failure message to the last successful hop (i.e., the one which provided the failed peer address). Upon receiving the failure message, the last hop tries to contact the failed finger entry, and if it fails again, it replaces the failed entry with the predecessor entry in the table. Furthermore, the peers maintain statistics about success and failures in the finger tables. If the failure count exceeds the success count by two or more, the corresponding entries are requested to check for failures and update if needed.

The two baseline stores, employing MR-Chord and Chord, use the same number of peers to store the content as MobiStore. They also use the same method to replicate the stored content. Furthermore, the lookup process retries three times before declaring a failure (i.e., the same with MobiStore). Therefore, the difference between MobiStore and MR-Chord or Chord comes only from the management and structure of the P2P network.

The evaluation has four goals. We quantify: (1) the resilience to churn and availability benefits; (2) the effects of scale on availability, latency, and overhead; (3) the benefits of load balance, and (4) the update latency, which can give application developers a better idea about the type of applications that are feasible over MobiStore.

Table 1 Simulation Parameters

Parameter	Range
Number of peers	6500
Peers per VP	2 - 25
Keys stored	2^{22}
Stored value size	10KB - 1MB
Lookup rate	2 - 3 per peer, per minute
Chord base	128
MR-Chord base	128
Chord finger update interval	4 minutes
Random links	10
Session time (ON)	2 - 60 minutes
Session time (OFF)	0 - 20 minutes
Packet processing and propagation delay	2 - 41ms
Avg WiFi bandwidth	54 Mbps
Avg cellular bandwidth	100 Kbps-10 Mbps
MobiStore global update interval	2 minutes
MobiStore local update interval	30 seconds
Load balance interval	20 minutes
Mobile peer speed	0.2 - 20 meters per second
Peers using cellular	30% time

5.1 Simulation Setup

Peers are initially placed randomly in a square region 40KM by 40KM. The maximum number of peers in our simulation is 6,500 due to the computation/memory limitations of PeerSim. The peers leave the network at exponentially distributed intervals with a session time ranging from 2 minutes to 1 hour. We use this limit to mimic the short session times of mobile devices. After completing an active session, the peers leave the P2P network for periods ranging from 0 to 20 minutes.

We used the BonnMotion [1] tool to generate mobility traces using the Gauss-Markov mobility model. This model maintains temporal dependency for modeling velocities and directions. For each peer initial position, the velocity and direction are chosen uniformly distributed over the simulation region (40KM by 40KM, which is divided in a 200,000 by 200,000 grid). Then the speed and direction of each peer is changed after an interval uniformly distributed between 0 and 60 seconds. The speed of the peers varies randomly in the range 0.2-20 meters per second. Each peer, on average, remains mobile for 30% of the simulation time and uses the cellular network to connect to the Internet and P2P network. The rest of the time, the peers connect to the Internet/P2P network using WiFi. In 80% of the grid cells, the peers can always connect to the cellular network and have an average bandwidth of 10 Mbps. In the other 20% of cells, the peers can connect with a probability of 0.8 and with varying bandwidth in the range of 100Kbps-10Mbps (chosen uniformly). The average bandwidth for WiFi-connected peers is assumed to be 54 Mbps. Although in real world the bandwidth values change based on different technologies, the important factor is the bandwidth ratio between WiFi and cellular network, not the absolute values.

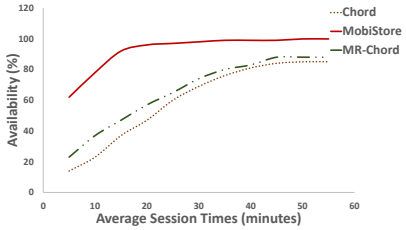

Fig. 4 Availability of MobiStore vs. Baselines as measured by lookup success rate

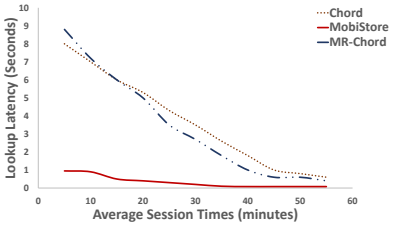

Fig. 5 Lookup latency of MobiStore vs. Baselines

The propagation, processing, and queuing delay are modeled together as a function of the corresponding peer resources and the Euclidean distance between the communicating peers. This includes wireless communication (last hop) and wired communication (i.e., between base stations/access points). Overall, the delay varies between 2ms and 41ms. The corresponding transmission delay (function of the packet size and bandwidth) is added to this value. The rest of the simulation parameters are listed in Table 1.

5.2 Comparison with MR-Chord and Chord Baselines

For this experiment, MobiStore has 930 VPs, each VP containing 7 peers. The Baselines have 6,510 peers (i.e., 930 x 7). The peers randomly generate and store 2^{22} key-value pairs before the simulation begins. Each peer issues lookups for random keys at intervals exponentially distributed with a mean of 20-30 seconds. For lookups, the peers only choose existing keys. The failures consist of routing failures, peer unavailability, or delayed replication.

Availability. Figure 4 shows the lookup success rate (which measures availability) for MobiStore vs. Baselines. MobiStore has a substantially higher availability than both MR-Chord and Chord Baselines, especially for shorter sessions. For example, MobiStore has 92% success rate for a 15-minutes session time, while MR-Chord has 47% and Chord has 37%. Also, MobiStore has high availability for sessions longer than 20 minutes. The results corroborate what we described in the design sections: well managed redundancy improves significantly the availability of MobiStore. MobiStore ensures that the requests are routed with fault-tolerance using multiple peers per VP and replicates the content among the members of the VPs. MR-Chord and Chord, on the other hand, suffer greatly from peer failures when it comes to propagation of routing information and content replication over multiple hops.

Latency. Figure 5 shows the latency of the successful lookups. MobiStore achieves a latency as low as 9 times the latency of MR-Chord (for 5 minutes session time, MR-Chord latency is 8.8 seconds and MobiStore latency is 0.95 seconds). This is due mostly to the 1-hop routing employed by MobiStore. We also observe that MobiStore's latency is acceptable for most mobile applications. Even for an average session of 15 minutes, the latency is as low as half

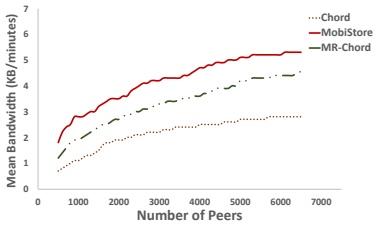

Fig. 6 Per peer management overhead traffic for MobiStore vs. Baselines

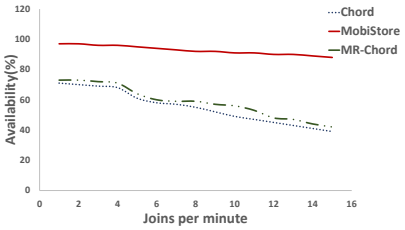

Fig. 7 Availability (measured by lookup success rate) as a function of parallel joins for MobiStore vs. Baselines

a second. This may not satisfy hard real-time constraints, but it is sufficient for most mobile applications.

We see that the lookup latency is higher for MR-Chord Baseline than for the Chord Baseline. The reason is MR-Chord increases the hop count during excessive churn while coping with failures. Therefore, for small session times, MR-Chord performs worse than Chord in terms of latency, but it increases the availability as found from Figure 4.

Overhead. The availability and latency benefits provided by MobiStore come at the expense of extra-overhead to maintain the network structure. MobiStore propagates aggregated routing tables which consume more bandwidth. For this experiment, MobiStore used a combination of full and "diff" updates of the aggregated routing tables. Every fourth update transmits the full table to help with consistency; in between, only the modified entries are transmitted. Figure 6 shows that MobiStore and MR-Chord have similar growth patterns for network management overhead. However, the absolute value of the overhead is low, each peer sending around 5KB per minute for a network of 6,500 peers. MobiStore uses around 1KB/minute more bandwidth than MR-Chord, but with this little extra overhead MobiStore can improve availability and decrease latency, which has the potential to reduce the number of data packet retransmissions and save bandwidth usage in future.

Scalability. Figure 6 also demonstrates that the growth in the overhead with the increase of peer count is between sub-linear and logarithmic. Since the update process is hierarchical and VPs have a limit on how many peer they can have, we conclude that MobiStore scales well with the increase in the number of peers: the more peers in the network, the closer to a logarithmic function the overhead is. Furthermore, Figures 7 and 8 show that MobiStore is able to maintain the availability benefits almost constant with the increase in new peer joins (Figure 7) or network size (Figure 8). On the other hand, availability decreases more rapidly with parallel joins for MR-Chord and Chord Baselines. These results hold even for high rates of parallel joins or larger network size. As we stated earlier, if a peer cannot find data in its own storage, it silently forwards the request to a fellow peer in the same VP. Therefore, the lookup succeeds with high probability if the data is present somewhere in the network for MobiStore. MR-Chord and Chord on the other hand, need the

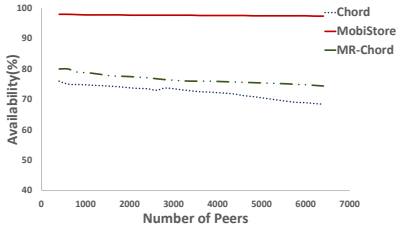

Fig. 8 Availability (measured by lookup success rate) as a function of network size for MobiStore

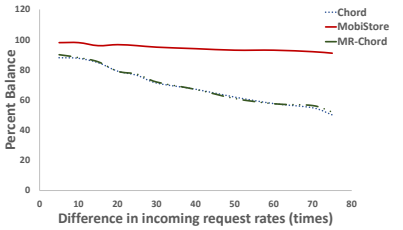

Fig. 9 Fraction of peers receiving almost the same load (±10% mean-load)

entire request forwarding path (all the routing hops) to work properly; this is affected adversely if new peers keep joining or the network size increases. Thus, based on our results, we conclude that MobiStore scales well to moderately large size networks.

5.3 Load Balance

For this experiment, we changed the content popularity for different stored values. The most popular content receives 5-75 times more requests than the least popular content. We used 325 VPs, with 20 peers each. The maximum number of peers per VP is 40 (after the VPs acquire additional peers), and the minimum peer number is fixed at 10. Therefore, 10 peers in each VP are loose peers (as described in the load balance discussion of Section 4). For MR-Chord and Chord, we used the average balance framework present in PeerSim [22]. The average balance methods tries to keep the load equal to average values over the peers. MobiStore uses the methods described in section 4.4.

Figure 9 presents the achieved load balance of MobiStore. The fraction of peers receiving $mean\_load \pm 10\%$ is higher than 91% in the worst case for MobiStore although the content popularity varies substantially. The rest of 9% peers experience 80% more load. For a similar load imbalance, MR-Chord or Chord Baselines can maintain only 50% peers receiving $mean\_load \pm 10\%$. Therefore, almost half of the peers receive significantly more load than others. This result demonstrates the benefits of integrated load balance techniques in MobiStore. All three techniques, namely, random request distribution, request forwarding, and changing the peer count in VPs are working to achieve this balance.

Figure 10 demonstrate the load balance process in more details as it shows the change in VP count and per-VP peer count as the incoming request rate increases. After applying the load balance techniques, we see that the peer count difference between VPs grows up to 30 (maximum possible in this setup). We also see from the figure that the load balance effort changed the number of VPs in the system from 325 to 225. The eliminated VPs are those which received the fewest requests, and they were merged with clockwise next VPs.

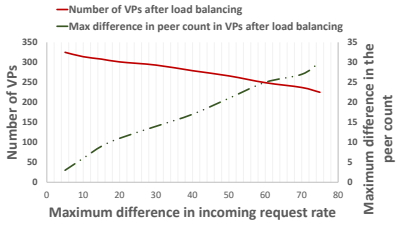

Fig. 10 Effect of load balance on the network (number of VPs and per-VP peer counts) as a function of request rate

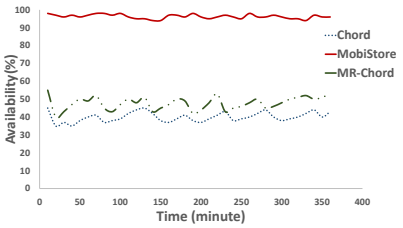

Fig. 11 Effect of load balance on availability over time (lookup success)

These results show that our load balancing techniques can quickly adapt to load variations.

Finally, Figure 11 presents the effect of load balance on availability over time. The availability fluctuates, but it is not grossly affected. Ideally, the load balance should not have any effect on availability because it does not affect the stored content. Failures occur, however, due the temporary inconsistency of the routing entries caused by moving peers from one VP to another. Nevertheless, the routing tables become consistent quickly and the availability improves.

5.4 Update Latency

These experiments assess both management update latency and content update latency. For both experiments, we randomly selected 10% peers to generate management data/new content. The selected peers post management data/new content in parallel. The size of management data depends on the routing table size, and the size of the content comes from a Pareto distribution with the shape parameter set to 0.5 and the mean parameter set to 100 (the numbers correspond to file sizes in KB). We selected the file sizes in the range 10 KB-1MB as typical size of the data stored such that the simulation results are not dominated by the file size; instead, they show the effect of the number of stored files. After every update is posted, 40 randomly selected peers over the whole system attempt to retrieve the management data/content. These 40 peers do not experience churn. We recorded the time of posting the updates and the time of successful retrieval of the content by all 50 peers. The time difference is the update latency.

Figure 12 shows that the management update latency is less than 4 minutes for most session times. For very short sessions, it grows up to 7 minutes. In fact, the latency depends on the periodic global update interval, which was fixed at 2 minutes in this experiment. Therefore, even in the worst case scenario, the updates reach everyone in just four global update intervals. This result demonstrates the robustness of the update process. Aggregated updates along with MobiStore structure and interval selection based on local clock helped to achieve this result.

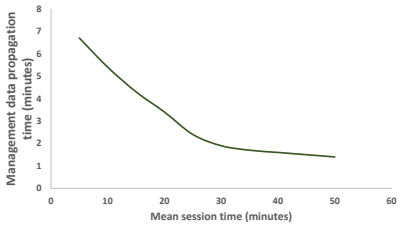

Fig. 12 Management data update latency as a function of session time for periodic updates

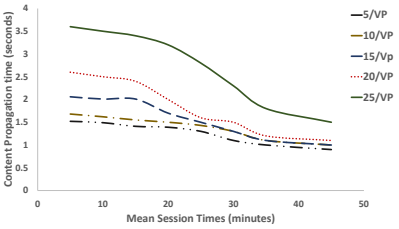

Fig. 13 Content update latency as a function of session time

Figure 13 shows the content update latency as a function of the session time; the content updates are asynchronous. Once the content is posted, MobiStore starts the update process. Only the peers currently using WiFi take part in the content update process. Therefore, the peers currently using cellular communication wait for the updates until they switch to WiFi. If a request for content comes to a cellular peer and the peer does not have the content, the peer forwards the request to a peer using WiFi (even without knowing anything about the content). Therefore VPs with 10-15 peers can find the content in 2-3 seconds in the worst case. Of course, larger file sizes would result in larger latency. Therefore, mobile applications are only limited by the wireless networking bandwidth. MobiStore does not increase the latency of the content update propagation; it rather minimizes the latency using silent forwarding among fellow peers of the VP.

6 Conclusion and Future Work

This paper presented MobiStore, a mobile P2P data store for sharing user-generated mobile content. While P2P techniques are well known and understood, there is currently no good P2P solution in the mobile world. The main reasons for this situation are short wireless sessions, i.e., very high churn, and resource limitations in terms of battery and mobility related issues. The MobiStore design addresses these constraints with a new mobile P2P network structure and mechanisms able to adapt to failures and load variation. The results demonstrate that MobiStore achieves high availability, low latency, and good load balance, without incurring high overhead that could impact negatively the performance in relatively large scale networks. MobiStore is ideal for applications which can tolerate worst case key-value update delays of several seconds.

As future work, we plan to investigate: (1) storage optimization techniques which may prove valuable if large amounts of data are stored in MobiStore; (2) quantifying the privacy benefits of MobiStore compared to a centralized solution, with a focus on location-based data sharing; and (3) providing security measures against personal information leakage attacks (e.g., location).

Acknowledgements This research was supported by the National Science Foundation (NSF) under Grants No. CNS 1409523 and DGE 1565478, and the National Security Agency (NSA) under Grant H98230-15-1-0274. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF and NSA. The United States Government is authorized to reproduce and distribute reprints notwithstanding any copyright notice herein.

References

1. A mobility scenario generation and analysis tool (2016). URL http://sys.cs.uos.de/bonnmotion/
2. Araujo, F., Rodrigues, L., Kaiser, J., Liu, C., Mitidieri, C.: Chr: a distributed hash table for wireless ad hoc networks. In: Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on, pp. 407–413. IEEE (2005)
3. Binzenhofer, A., Staehle, D., Henjes, R.: On the stability of chord-based p2p systems. In: Global Telecommunications Conference, 2005. GLOBECOM'05. IEEE, vol. 2, pp. 5–pp. IEEE (2005)
4. Caesar, M., Castro, M., Nightingale, E.B., O'Shea, G., Rowstron, A.: Virtual ring routing: network routing inspired by dhts. In: ACM SIGCOMM Computer Communication Review, vol. 36, pp. 351–362. ACM (2006)
5. Cao, H., Wolfson, O., Xu, B., Yin, H.: Mobi-dic: Mobile discovery of local resources in peer-to-peer wireless network. IEEE Data Eng. Bull **28**(3), 11–18 (2005)
6. Cao, Q., Fujita, S.: Load balancing schemes for a hierarchical peer-to-peer file search system. In: P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2010 International Conference on, pp. 63–70. IEEE (2010)
7. Chakravorty, R., Agarwal, S., Banerjee, S., Pratt, I.: Mob: A mobile bazaar for wide-area wireless services. In: Proceedings of the 11th annual international conference on Mobile computing and networking, pp. 228–242. ACM (2005)
8. Druschel, P., Rowstron, A.: Past: A large-scale, persistent peer-to-peer storage utility. In: Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on, pp. 75–80. IEEE (2001)
9. Erd6s, P., Rényi, A.: On the evolution of random graphs. Publ. Math. Inst. Hungar. Acad. Sci **5**, 17–61 (1960)
10. Fitchard, K.: Can cell phone data cure society's ills? (2012). URL http://gigaom.com/2012/03/11/10-ways-big-data-is-changing-everything/8/
11. Gupta, I., Birman, K., Linga, P., Demers, A., Van Renesse, R.: Kelips: Building an efficient and stable p2p dht through increased memory and background overhead. In: Peer-to-Peer Systems II, pp. 160–169. Springer (2003)
12. Hofstatter, Q., Zols, S., Michel, M., Despotovic, Z., Kellerer, W.: Chordella-a hierarchical peer-to-peer overlay implementation for heterogeneous, mobile environments. In: Peer-to-Peer Computing, 2008. P2P'08. Eighth International Conference on, pp. 75–76. IEEE (2008)
13. Horozov, T., Grama, A., Vasudevan, V., Landis, S.: Moby-a mobile peer-to-peer service and data network. In: Parallel Processing, 2002. Proceedings. International Conference on, pp. 437–444. IEEE (2002)
14. Jiang, W., Xu, C., Huang, M., Lai, J., Xu, S.: Improved chord algorithm in mobile peer-to-peer network (2011)
15. Karthik, B.G., Lakshminarayanan, K., Surana, S., Karp, R., Stoica, I.: Load balancing in dynamic structured p2p systems. In: In Proc. IEEE INFOCOM, Hong Kong. Citeseer (2004)
16. Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., et al.: Oceanstore: An architecture for global-scale persistent storage. ACM Sigplan Notices **35**(11), 190–201 (2000)
17. Landsiedel, O., Lehmann, K.A., Wehrle, K.: T-dht: topology-based distributed hash tables. In: Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on, pp. 143–144. IEEE (2005)

18. Lee, J.W., Schulzrinne, H., Kellerer, W., Despotovic, Z.: mdht: multicast-augmented dht architecture for high availability and immunity to churn. In: Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE, pp. 1–5. IEEE (2009)
19. Lee, U., Lee, J., Park, J.S., Gerla, M.: Fleanet: A virtual market place on vehicular networks. Vehicular Technology, IEEE Transactions on **59**(1), 344–355 (2010)
20. Liquori, L., Tedeschi, C., Vanni, L., Bongiovanni, F., Ciancaglini, V., Marinković, B.: Synapse: A scalable protocol for interconnecting heterogeneous overlay networks. In: NETWORKING 2010, pp. 67–82. Springer (2010)
21. Liu, C.L., Wang, C.Y., Wei, H.Y.: Cross-layer mobile chord p2p protocol design for vanet. International Journal of Ad Hoc and Ubiquitous Computing **6**(3), 150–163 (2010)
22. Montresor, A., Jelasity, M.: PeerSim: A scalable P2P simulator. In: Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09), pp. 99–100. Seattle, WA (2009)
23. Mordacchini, M., Ricci, L., Ferrucci, L., Albano, M., Baraglia, R.: Hivory: Range queries on hierarchical voronoi overlays. In: Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on, pp. 1–10. IEEE (2010)
24. Orf, D.: So whatever happened to post-pc? (2014). URL http://www.gizmodo.in/gadgets/So-Whatever-Happened-to-Post-PC/articleshow/40069551.cms
25. Pásztor, B., Musolesi, M., Mascolo, C.: Opportunistic mobile sensor data collection with scar. In: Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE Internatonal Conference on, pp. 1–12. IEEE (2007)
26. Rao, A., Lakshminarayanan, K., Surana, S., Karp, R., Stoica, I.: Load balancing in structured p2p systems. In: Peer-to-Peer Systems II, pp. 68–79. Springer (2003)
27. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '01, pp. 161–172. ACM, New York, NY, USA (2001)
28. Ratnasamy, S., Karp, B., Yin, L., Yu, F., Estrin, D., Govindan, R., Shenker, S.: Ght: a geographic hash table for data-centric storage. In: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications, pp. 78–87. ACM (2002)
29. Rhea, S., Geels, D., Roscoe, T., Kubiatowicz, J.: Handling churn in a dht. In: Proceedings of the USENIX Annual Technical Conference, pp. 127–140. Boston, MA, USA (2004)
30. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Middleware 2001, pp. 329–350. Springer (2001)
31. Rybicki, J., Scheuermann, B., Koegel, M., Mauve, M.: Peertis: a peer-to-peer traffic information system. In: Proceedings of the sixth ACM international workshop on VehiculAr InterNETworking, pp. 23–32. ACM (2009)
32. Shaker, A., Reeves, D.S.: Self-stabilizing structured ring topology p2p systems. In: Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on, pp. 39–46. IEEE (2005)
33. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. Networking, IEEE/ACM Transactions on **11**(1), 17–32 (2003)
34. Waal, M.: Mobile phones, social networks and location data: Recognizing the nuances of privacy (2010). URL http://www.themobilecity.nl/2010/06/10/mobile-phones-social-networks-and-location-data-recognizing-the-nuances-of-privacy/
35. Woungang, I., Tseng, F.H., Lin, Y.H., Chou, L.D., Chao, H.C., Obaidat, M.S.: Mr-chord: Improved chord lookup performance in structured mobile p2p networks (99), 1–9 (2014)
36. Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.D.: Tapestry: A resilient global-scale overlay for service deployment. Selected Areas in Communications, IEEE Journal on **22**(1), 41–53 (2004)
37. Zulhasnine, M., Huang, C., Srinivasan, A.: Towards an effective integration of cellular users to the structured peer-to-peer network. Peer-to-Peer Networking and Applications **5**(2), 178–192 (2012)