GridTop: Grid Top-k Query with Location-based Replication in Mobile Ad Hoc Networks

Yuya Sasaki #1, Takahiro Hara #1 and Nishio Shojiro #1

Email: { sasaki.yuya, hara, nishio} @ist.osaka-u.ac.jp
#1*Osaka University, Yamadaoka 1-1, Suita-shi, Osaka, 565-0871, Japan.*

***Abstract*—In this paper, we propose a method which combines top-k query processing and location-based replication in MANETs called $GridTop$ (grid-based top-k query processing with location-based replication) to achieve small overhead and delay. In this method, the entire area is divided into grids based on the buffer size and the communication range of nodes, and the data items are then allocated to nodes in the grids. Since data items are geographically and uniformly allocated, a query issuer can identify the grids where nodes have the necessary data items. In GridTop, the query issuer only sends a query message to necessary grids to obtain top-k result. When each node has replicas, the same data items probably are sent back to the query issuer. To prevent it, each node attaches the data identifiers whose data items have the k highest score held by a node to the query message. As a result, GridTop achieves the small overhead and delay, and also keeps high accuracy of query result. The simulation result shows that GridTop achieves high performance.**

***Keywords*-Top-k query, Replication, Mobile ad hoc network**

I. INTRODUCTION

There has recently been an increasing amount of interest in $mobile\ ad\ hoc\ networks$ ($MANETs$) in which all mobile nodes play the role of routers and communicate with one another. Even if the source and destination nodes are not within the communication range of each other, data packets are forwarded to the destination node by relaying the transmission through other nodes that are between the two nodes. Since a special infrastructure is not required, in various fields such as military affairs and rescue operations, many applications are expected to be developed for MANETs. However, technical problems become complex as nodes in MANETs move freely, and their batteries, their computational resources, and their communication bandwidths are limited.

It is important for query processing to only acquire necessary data items with small overhead and delay because of these issues. Possible and promising solutions are $top\text{-}k\ queries$ and $replication$. In a top-k query, data items are ordered by the score of particular attributes and the query issuer acquires data items with the k highest scores ($top\text{-}k\ result$). In general replication, the most popular data items are allocated to each node to improve the data availability. Location-based replication can especially allocate data items uniformly in the assumed area. The overhead and delay can be reduced by using these solutions.

A large number of replication strategies have been proposed in the literature for fixed and mobile networks. Data items with high access frequency have often been allocated in these strategies. However, these strategies are not suitable for top-k queries because they assumed that a query accesses only one data item, and too many of the same data items are allocated. In a top-k query, since each node retrieves k data items, the access frequency of a data item with the highest score is 100% and that of data items with a very high score may also be almost 100%. However, in most of the existing strategies, a query message is sent to all nodes in the network in order to accurately acquire the top-k result. As a result, too many same data items are sent back to the query issuer, which increases the overhead and delay.

To address the above issues, we have two main problems. First, as k and the buffer size are interdependent, they should be considered together. Since a query message should not be sent to all nodes in the entire network, the number of hop counts from the query issuer should be restricted, and also the top-k result must be guaranteed. The optimal replication should be achieved to reduce overhead and delay, but this problem is known to be NP hard. Second, reallocation occurs due to the movement of nodes and the deletion/generation of data items. The overhead should be small when data items need to be reallocated. We should consider the initial allocation of data items to reduce the opportunities for reallocation and how data items are reallocated to achieve a small overhead.

In this paper, we propose a method which combines top-k query processing and location-based replication in MANETs called $GridTop$ (grid-based top-k query processing with location-based replication) to achieve a small overhead and delay. In this method, the entire area is divided into grids based on the buffer size and the communication range, and data items are then allocated to nodes in the grids. Since data items are geographically and uniformly allocated, a query issuer can identify grids where nodes have the necessary data items. In particular, when k is small, the query issuer sends a query message to nodes in only its own grid. We

consider three cases of reallocation. The first is where a node leaves the grid, and all replicas that this node has normally need to be reallocated. However, as this involves a large overhead, if the out-migrant node has data items that many nodes in the entire network have, reallocation is not done at this timing. This is because even if some data items in its own grid are missing, there are data items in a nearby grid that can be reallocated. On the other hand, data items that only a small number of nodes have should be reallocated because chances for reallocation are small. The second case is that data items are deleted. When a node deletes data items with high scores, it sends the notification to all grids and nodes having replicas of these data items also delete them. As the ranking of scores changes, replicas are dynamically reallocated. However, as this also entails large overhead, the initial allocation is designed so that reallocation is not needed even if some data items are deleted. The third case is where data items are generated and new data items must be sent to the nodes related to them. Those data items are sent to the all grids to notify changing the ranking and are reallocated. In GridTop, the query issuer only sends a query message to necessary grids to obtain the top-k result. When each node has replicas, the same data items are probably sent back to the query issuer. To prevent this, each node attaches data identifiers whose data items have the k highest score held by the node to a query message. As a result, GridTop achieves small overhead and delay, and also keeps high accuracy of query result.

The contributions of this paper are as follows:

- The proposed approach is the first method which combines top-k query processing and location-based replication in MANETs.
- We formulate location-based replication where the query issuer can acquire the correct top-k result by using a restricted area and the opportunities for reallocation are small.
- The query issuer in a top-k query determines the search range based on k. This achieves small overhead and delay, and also the top-k result is guaranteed.
- We show that our protocols worked well in terms of small delay and traffic through extensive simulations that take into account the effect of the physical layer.

The remainder of this paper is organized as follows: Section II introduces related work. Section III explains our system model. Section IV outlines the problem statements and Section V presents our proposed method. Section VII provides the results obtained from the simulation experiments. Finally, Section VIII summarizes the paper.

II. Related work

Several protocols for caching and data replication have been proposed in MANETs and top-k query processing in several networks. We introduce some important protocols.

A. Caching and Replication

First, we introduce some $Cooperative\ Caching$ protocols in MANETs. In [11], the authors proposed three schemes of CacheData, CachePath and HybridCache as cooperative caching techniques to efficiently support data access. In CacheData, intermediate nodes cache data items to utilize for future requests. In CachePath, nodes cache the data path instead of data items and use this to forward future requests to a nearby node that has the data items. HybridCache combines CacheData and CachePath and caches data items if they are small, otherwise it caches the data path. In [10], the authors proposed a cache placement algorithm to reduce the total access cost with limited memory capacity. In [4] and [5], the authors proposed cooperative caching called Hamlet that was aimed at creating diversity within neighboring nodes so that users would likely find required data items nearby and avoid flooding the network with query messages. In [5], the caching decisions were separately taken by each node, and only took into account content query and reply messages. In [4], the caching decisions also took into account the size of the cache. In [3], the authors proposed a centralized contention-aware caching algorithms (CCCA) and a distributed contention-aware caching algorithm (DCCA) that aimed at reducing access traffic flows and increasing update traffic flows.

Second, we introduce some $Data\ Replication$ protocols in MANETs. In [8], the author proposed three algorithms for replica placement to improve the data availability. In the first approach, each node caches data items based on access frequency by itself. The second algorithm extends the first approach in that no nodes cache data items that neighboring nodes have as a replica. The third algorithm takes into account the network topology. In [9], the authors proposed location-based replication called a location-aided content management architecture (LACMA) that was aimed at finding the required data items from specific areas with a high probability of being located there. In LACMA, data items are allocated to a specific grid based on access frequency and data items with high access frequency are specifically allocated to small grid sizes. Nodes determine whether to push the data items or not when the leaving current grid. If the grid contains at least one replica with high probability, it does not push the data items.

These protocols basically assume that each node requests one data item with one query; however, a rank-based query like a top-k query retrieves several data items with one query. As our proposed protocol aims at acquiring k data items through small hop counts without flooding the entire network with queries, it is more difficult to replicate data items to reduce cost. Moreover, all the protocols basically do not take into consideration the deletion and update of data items. We not only take into consideration the movement of nodes but also the deletion and update of data items.

Finally, in our previous work [7], we proposed a replication method for top-k query processing, where each node allocates data items when it receives and relays the top-k query result. As each node allocates data items based on the score and a random number, data items with high scores are often allocated, but the replicas also maintains diversity in data items. However, as this method does not take into consideration top-k query processing after allocation, many of the same data items are replied, and this replication is far from being optimal.

B. top-k query processing

A large number of methods of top-k processing in several kinds of networks have been proposed.

Many strategies for top-k queries have been proposed [1], [2], [12], [14], [15] in the research field of fixed P2P networks. In [15], the authors proposed a method in which each node that receives a top-k query message orders its own data items by score, and transmits data items with the k highest scores. In [2], the authors proposed a method in which a query-issuing node acquires the data item with the highest score by transmitting a query and it continues this procedure k times to acquire the top-k result. In [1], the authors proposed a method in which each mobile node selectively transmits query and reply messages to reduce network traffic. In [12], the authors proposed a method in which each node determines the number of data items included in a reply message based on the number of its children and k, and sends back the determined number of data items with the highest scores. In [14], the authors proposed a method in which super peers collect data items with high scores from their children nodes, and only super peers communicate with one another. The methods proposed in [2] and [15] do not work well in MANETs because the former method causes very large traffic and the latter causes query processing delay. The method proposed in [12] cannot be directly applied to MANETs because no nodes can know the number of its children in query transmissions in advance. The method proposed in [1] cannot be applied to MANETs because it assumes an environment where every message is transmitted by unicasting and every two nodes can communicate with each other at anytime. The method proposed in [14] can reduce traffic due to limited communication between super peers, but it is difficult to appropriately select super peers in MANETs. Moreover, if disconnection between super peers occurs, the accuracy of the query result drastically decreases.

To the best of our knowledge, our previous work [6] and the method proposed in [13] are the only existing studies that address the issue of top-k query processing in MANETs. However, in [6] we did not assume data replication. The method in [13] assumed an economic scheme (e.g., virtual currency) in which their assumption was quite different from ours.

III. System model

The system environment is assumed to be a MANETs in which mobile nodes retrieve data items held by themselves and other mobile nodes using a top-k query. The query issuer transmits a query message with the query condition and the number of data items, k, and it then acquires data items with the k highest scores of all data items held by nodes in the entire network. The number of k is chosen from 1 to k_{max}.

We assign a unique $data\ identifier$ to each data item in the system. The set of all data items in the system is denoted by $d = \{d_1, d_2, \cdots, d_D\}$, where D is the total number of data items and $d_i (1 \leq i \leq D)$ is a data identifier. We define the subscription of d as the score rank (i.e., d_1 has the highest score and d_k has the k highest score). Each data item is held by a specific node. For simplicity, all data items are the same size. The scores of data items can be calculated from the query condition and various scoring functions. The data items are deleted and generated as time passes.

We assign a unique $node\ identifier$ to each mobile node in the system. The set of all nodes in the system is denoted by $m = \{m_1, m_2, \cdots, m_M\}$, where M is the total number of nodes and $m_i (1 \leq i \leq M)$ is a node identifier. Each mobile node moves freely. Each node can allocate B data items as replicas. For simplicity, B is the same size for each node. Every mobile node recognizes its own location established by GPS.

IV. Problem statement

A. Replication problem

The replication problem in the entire network to achieve optimal allocation is known to be NP-hard. The total cost in optimal allocation to acquire k data items, T, becomes the smallest value. The query issuer, x, designates the number of requested data items, k ($\leq k_{max}$), and it then acquires data items, d_i ($1 \leq i \leq k$). T can be calculated by the following equation:

$$T = \Sigma_{x \in m} \Sigma_i^{k_{max}} P(i) \Sigma_{j=1}^{i} d(x, d_j) \quad (1)$$

where $P(i)$ is the probability that i is designated k, and $d(x, d_i)$ is the number of hop counts from node x to the node that has d_j.

As this equation is based on the conventional approach for only one data access, one data item is sent back by one query. Where this approach is applied to a top-k query, the delay increases because it needs to send a query message k times. It is efficient to acquire k data items with one query. As the delay here is calculated based on the longest hop count, the longest hop count should be short. Here, we define the following new equation to calculate the total cost.

$$
\begin{aligned}
&T_{k_0}(x) = 0 \\
&T_{k_1}(x) = \Sigma_{i=k_0+1}^{k_1} P(i) \cdot (i - k_0) \\
&T_{k_2}(x) = \Sigma_{i=k_1+1}^{k_2} P(i) \cdot ((i - k_1) \cdot 2 + k_0) \\
&T_{k_n}(x) = \Sigma_{i=k_{n-1}+1}^{k_n} P(i) \cdot \\
&\qquad ((i - k_{n-1}) \cdot area + \cdots + k_1) \\
&T = \Sigma_{x \in m} \Sigma_{i=0}^{k_{max}} T_{k_i}(x) \qquad (2)
\end{aligned}
$$

where k_i $(0 \leq i \leq n$, and $k_i < k_{i+1})$ denotes that the query issuer acquires data items with the k_i highest scores by i hop counts (i.e., k_0 denotes that the query issuer has the data items with k_0 highest scores), $T_{k_i}(x)$ is the cost when x designates k between k_i and $k_{i+1} - 1$, and $area$ denotes the maximum hop counts where x transmits a query throughout the network.

This equation is used to calculate the total cost, but it does not take into consideration the buffer sizes of nodes. Cooperative replication in all nodes is necessary to achieve minimum total cost. Hence, we address the issues of determining replication that achieves a short hop count to acquire k data items and hop counts that are determined based on k. This is possible by using location-based replication, but it is a complex problem to resolve these issues.

B. Reallocation problem

Even when optimal allocation is achieved, that immediately changes due to the movement of nodes and the deletion and update of data items. This subsection discusses the cost of reallocation.

1) Movement of nodes: Since nodes move freely, it is difficult to maintain replicas with specific locations. When nodes move to a distant location, replicas should be sent to other nodes closer to points where replicas should be maintained. If the locations allocated to data items are small areas, the movement of nodes causes large overhead in reallocating data items. The cost of reallocation due to movement, $cost_{mov}$, is calculated by the following equation.

$$
cost_{mov} = \Sigma_{x \in m} L(x) \cdot B \qquad (3)
$$

where $L(x)$ denotes the probability that node x leaves a permissible area, and B denotes the buffer size of nodes. $L(x)$ is calculated based on the mobility model and the velocity of the node, and the size of the permissible area.

The permissible ares should be large to reduce $cost_{mov}$, but necessary data items in a top-k query should be acquired from nearby nodes.

2) Deletion/generation of data items: When data items are deleted and generated, a message that informs data items have been deleted should be sent to nodes that have the data items and the new data items should be sent to nodes that should be allocated. Since rankings change, reallocation is necessary. The cost of deletion, $cost_{del}$ and that of generation, $cost_{gen}$, are calculated with the following equations.

$$
cost_{del} = delete(x) \cdot (delete_{num} + \Sigma_{x \in m} reallocate). \quad (4)
$$

$$
cost_{gen} = generate(x) \cdot (generate_{num} + \Sigma_{x \in m} reallocate). \quad (5)
$$

where $delete(x)$ and $generate(x)$ correspond to the probability that deletion and update occur, $delete_{num}$ and $generate_{num}$ correspond the number of nodes that should send a message and data items, and $reallocate$ is the overhead of reallocation associated with changing the rank of data items n the entire network.

The first term in the equation of $cost_{del}$ can be ignored because only a message is sent instead of a data item, and also the first term in the equation of $cost_{gen}$ is smaller than the second term because only one data item is sent. Therefore, as $reallocate$ is dominant, it is important to ensure that the reallocation caused by the deletion and generation of data items is as small as possible.

C. Summarization of problems

Optimal allocation is calculated based on various factors, e.g., the buffer size, the speed of movement of nodes, the number of nodes, and the frequency of deletion and generation of data items. There are not only stable but also dynamical features in these factors. Moreover, it does not mean much even if optimal allocation can be achieved, but reallocation cost increases.

Therefore, as the total optimal allocation that takes into account reallocation is not easy to solve, we propose a practical approach.

V. Location-based replication

This section explains our proposed replication method. First, we describe the design policy and discuss the practical approach. Then, we describe how to replicate data items on each node.

A. Design policy

A top-k query should achieve a high delivery ratio, small delay, and small overhead. As replication can effectively help this, each node stores data items with high access frequency. Data items with high access frequency in a top-k query are high ranking data; in particular, the access frequency of data items with the highest score is 100%. High ranking data items are frequently allocated on numerous nodes. However, this is not effective because the buffer size of nodes is limited. Therefore, it is important for diverse data items and also the query issuer to receive the necessary data items from nearby nodes. We propose location-based replication where each node can acquire the top-k result

through the limited scope of its search for any k. Data items with high scores are actively allocated to the specific circular areas (i.e., $circular\ grids$) based on the discussion described in Section V-B. In our proposed top-k query processing method, as each node retrieves data items from nodes in circular grids that are determined based on k, the entire area does no need to be flooded. As a result, our proposed top-k achieves a high delivery ratio, small delay, and small overhead.

B. Discussion

The location-based replication we propose makes it possible to acquire data items with the k highest scores by using the small average hop count. First, we assume that grids whose radius is half of communication range cover the area and are deployed next to neighboring inscribed hexagons (see Figure 1). The average hop count, H_{avg}, is calculated by the following equation.

$$H_{avg} = \Sigma_{i=1}^{k} \alpha_i \cdot (2i-1) \tag{6}$$

$$c/k_{max} = \Sigma_{i=1}^{k} \frac{\alpha_i}{6 \cdot i - 5} \tag{7}$$

$$\Sigma_{i=1}^{k} \alpha_i = 1 \tag{8}$$

where c denotes the buffer size in a circular grid whose radius is half of R, and α_i denotes that a node can acquire the data items with $k_{max} \cdot \alpha_i$ highest scores in $(2i-1)$ hops. It is important to find α_i to achieve the smallest H_{avg}.

We present an almost optimal α_i for five case of c/k_{max}.

- $c/k_{max} \geq 1$ (i.e., it can be allocated more than k_{max} data items): when $\alpha_1 = 1.0$, the smallest H_{avg} is 1.
- $c/k_{max} = 0.75$: when $\alpha_1 = 0.7$ and $\alpha_2 = 0.3$, the smallest H_{avg} is 1.6.
- $c/k_{max} = 0.5$: when $\alpha_1 = 0.4$ and $\alpha_2 = 0.6$, the smallest H_{avg} is 2.2.
- $c/k_{max} = 0.25$: when $\alpha_1 = 0.12$ and $\alpha_2 = 0.88$, the smallest H_{avg} is 2.76.
- $c/k_{max} = 0.1$: when $\alpha_1 = 0$, $\alpha_2 = 0.5$ and $\alpha_3 = 0.5$, the smallest H_{avg} is 4.

These examples demonstrate that a node can acquire k_{max} data items by retrieving its own circular grid or the circular grids around its own except where c/k_{max} is too small. As the case where c/k_{max} is too small (e.g., $k_{max} = 10$ and $c = 1$) is not realistic, we have not considered it in this paper.

However, when the radius of the circular grid is $2 \cdot R$, the number of data items allocated in the grid is quadrupled, so the size should be large for $H_{avg} \geq 2$. Moreover, as $P(i)$ has been ignored in this equation, it is sometimes not necessary to change the size of the grid for $H_{avg} \geq 2$. For instance, if each node for $c/k_{max} = 0.5$ designates a value less than $0.4 \cdot k_{max}$ as k with a probability of more than 0.5, H_{avg} becomes less than two.

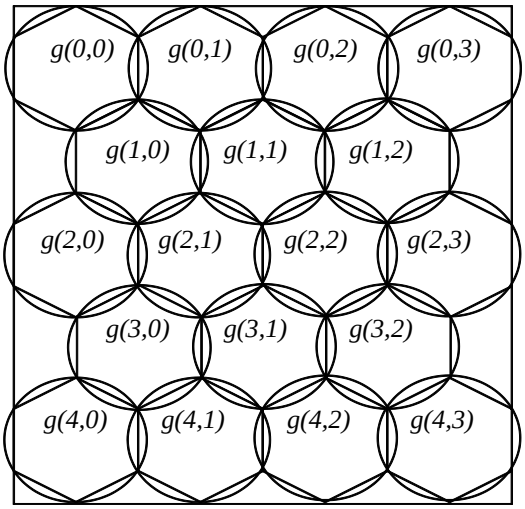


Figure 1. Grid deployment

We aim at a node acquiring the requested data items by retrieving them from only its own circular grid with a probability of more than 0.5 and acquiring data items with k_{max} highest scores by retrieving the circular grids around its own grid under these conditions with the location-based replication we propose.

C. Initial Allocation

The above discussion pointed out that each node can acquire the data items with the k_{max} highest scores by retrieving its own grid and grids around its own.

First, let us describe the deployment of grids. The grids, $g(i,j)$, cover the area and are deployed next to neighboring inscribed hexagons (see Figure 1). The i and j in the grid identifier, $g(i,j)$, correspond to the column and the row numbers. Here, the grid does not tidily cover the rectangular area, and some nodes do not belong to any grids.

Each node determines what data items are allocated based on the grid that it belongs to. The data items are allocated with the following rules:

$$\Sigma_{i=1}^{k_1} P(i) \geq 0.5 \tag{9}$$

$$c_k = k_1 + \lceil (k_{max} - k_1)/7 \rceil + margin \tag{10}$$

where k_1 denotes that each node can acquire the data items with the k_1 highest score by retrieving its own grid (k_1 is calculated by the probability that each node designates i as k), c_k denotes the number of data items that each grid should allocate, and $margin$ denotes the number of margins that each node sets to allocate extra data items to avoid reallocation. (For example, when $margin$ is 5, the reallocation is unnecessary if 5 data items are continuously deleted.) The grid, $g(i,j)$, basically allocates data items from the highest scores through k_1 highest scores, and from $(k_1 + \lceil \frac{k_{max}-k_1}{7} \rceil \cdot (\lfloor \frac{i+5 \cdot (j \bmod 2)}{2} \rfloor - 2 \cdot (j \bmod 2)) \bmod 7 + 1)$

through $(k_1 + \lceil \frac{k_{max}-k_1}{7} \rceil \cdot (\lfloor \frac{i+5\cdot(j \bmod 2)}{2} \rfloor - 2 \cdot (j \bmod 2)) \bmod 7 + \lceil \frac{k_{max}-k_1}{7} \rceil)$ highest scores. Here, $margin$ expands the range of replicas, so even if some data items are deleted and generated, reallocation does not occur. Nodes that do not belong to a grid allocate the B highest scores, which they rely on nearby grids in a top-k query.

The size of a grid is calculated by the number of data items that each grid can allocate, $c_{g(i,j)}$. All $c_{g(i,j)}$ must be more than c_k. When above rules cannot be followed, the size of grid increases. (In particular, the grid size is $i \cdot R \ \ (i = 1, 2, \cdots)$ until only one grid covers all the area.) If some grids can allocate more data items than c_k, they can allocate data items with the highest scores possible to acquire the data items from their own grids.

In this allocation, each node retrieves the necessary data items from their own grids with more than 50% probability. Moreover, even if $c_{g(i,j)}$ is variable, the average hop count can decrease.

For example, where k_{max} is 100 and k_1 is 50, c_k becomes 58. Therefore, $c_{g(i,j)}$ have to be more than 58. For instance, $g(0,0)$ allocates the data items from first through to the 50-th highest scores, and from 51-st through to the 58-th scores.

D. Allocation of data items

We adopted the centralized approach in allocating data items.

First, a node ($coordinator$) collects the data items with the $k_{max} + margin$ highest scores and the positions of nodes by using flooding. Then, the coordinator determines the grid size based on the positions of nodes. The coordinator transmits the data items, the position of nodes, and the grid size to all nodes. Each node that receives that information determines the replicas based on its own position and the positions of nodes in its own grid. When the node is closer to the center of the grid than other nodes, it allocates more important data items. This priority is itemized as follow.

- Top priority data: Data items from $(k_1 + \lceil \frac{k_{max}-k_1}{7} \rceil \cdot (\lfloor \frac{i+5\cdot(j \bmod 2)}{2} \rfloor - 2 \cdot (j \bmod 2)) \bmod 7 + 1)$ through $(k_1 + \lceil \frac{k_{max}-k_1}{7} \rceil \cdot (\lfloor \frac{i+5\cdot(j \bmod 2)}{2} \rfloor - 2 \cdot (j \bmod 2)) \bmod 7 + \lceil \frac{k_{max}-k_1}{7} \rceil)$ highest scores.
- Second priority data: Data items with more than k_1 scores.
- Margin data: Data items for $margin$.
- Other data: Other data items (but, more than $k_{max} + margin$ highest scores).

The accuracy of query results can be guaranteed by allocating data items according to the priority. The reason the highest score is second priority is that many grids have allocated these data items. In this scheme, each node knows how many replicas are deployed in its own grid, so in the top-k query processing it can determine the small search area to acquire the top-k result.

For example, observe the top priority, second priority and margin data where k_{max} is 100, k_1 is 50, and $margin$ is 10. The top priority data, second priority data and margin data for $g(0,0)$ are correspond to (51-st $\sim$ 58-th highest), (the first $\sim$ 50-th highest), and (59-th $\sim$ 68-th) scores. These data for $g(0,1)$ are correspond to (59-th $\sim$ 66-th highest), (the first $\sim$ 50-th highest), and (51-st $\sim$ 55-th and 68-th $\sim$ 72-nd) scores .

E. Reallocation

We discuss about the reallocation in this section. In the first allocation of data items, each node knows the information of its own grid (how many nodes, how data items, how many data items, and how many buffer spaces are available in its own grid). Each node keeps this information whenever the reallocation occurs. We explain the procedure for three cases of reallocation.

1) Data deletion: When a node deletes data items with $k + margin$ highest scores, it sends a $deletion\ message$ on the network to notice the deletion of data items. The deletion message is attached identifier of the deleted data item and the time that the node deletes data items. A node that received the message changes the score ranking. Then, if it has the deleted data items as a replica, it deletes the replicas and stores the time that the node deletes data items. Additionally, it changes the information of its own grid. Concretely, it changes the number of replicas in its own grid and registers how many available space are generated at this time. Even if some data items are deleted, the top-k result is guaranteed and the reallocation is not needed due to the margin. Each node knows the change of score ranking and keeps the information of its own grid with this procedure, and it takes no overhead for the reallocation.

2) Data generation: When a node generates data items with $k + margin$ highest scores, it sends a $generation\ message$ on the network to notice the generation of data items. The generation message is attached new data item. A node that received the message the changes the score ranking. The reallocation is taken into account the information of its own grid. There are four patterns to reallocate the data items. First, if there are replicas that becomes out of the score ranking in its own grid, nodes exchange that data item and new one. Second, there are nodes that have available space to allocate replicas in its own grid, nodes that have available space which generates at the oldest time in its own grid allocate the new data item. Third, nodes exchange the data items with the lowest priority and new data item. Fourth, if the new data items is the lowest priority in its own grid, it is not allocated in this time. Since this reallocation determines by the information that each node has, it does not need to exchange a message and data items among nodes. In the case that each node needs to reallocate, it keeps the information of its own grid.

3) Movement of nodes: Reallocation is needed when a node leaves its own grid. A node notifies nodes in the source

grid that it is leaving and nodes in the destination grid that it is participating, respectively.

The node sends a $moving\ message$ to neighboring nodes when it leaves its own grid. The moving message is attached to replicas that should be reallocated (Top priority data that is allocated one replicas in its own grid), data item identifiers whose replicas it has (except for attached top priority data), the available space information on this nodes (the time that it deletes the replicas), and grid identifiers of the source and destination grids. A node that received the message in the source grid and that has available space that generates at the oldest time in the grid or lowest priority data items in the grid allocates the received replicas. A node that received the message in the destination grid removes data items attached to the message to reduce the message size and it attaches the information of its own grid to notify the information to moving node after updating the information of the grid. Both nodes in the source and destination grids transmit messages to their neighboring nodes when the grid size is more than $2 \cdot R$. Each node can know the information of its own grid with this procedure.

VI. Top-k query processing

In GridTop, the query issuer can acquire data items by restricting the search range due to replication. As each node knows the allocated replicas in its own grid, it can determine what grids should be searched.

The query issuer first checks the allocated replicas with the k highest scores in its own grid. When nodes in its own grid have all the necessary data items, the query issuer sends a query message to those nodes in its own grid. However, if nodes in its own grid do not have some necessary data items, the query issuer sends a query message to nodes in nearby grids that allocate those data items. Here, we have assumed that all grids have at least allocated top priority data.

The procedures to transmit query and reply messages are as follows.

- The query issuer, m_p, specifies the number of requested data items, k, and the query condition. Then, m_p calculates the grids that it should search, and it transmits a query message to its neighboring nodes. The query message includes the query issuer ID, the query message ID, the source node ID, the parent node ID, the query condition, k, hop count from the query issuer, the grid ID list, and the data ID list. The grid ID list includes its own grid and other grids that should be searched to obtain the necessary data items, and the data ID list includes data item identifiers whose data items have the k highest scores held by m_p. The data ID list prevents that many of the same data items from being replied to the query issuer.
- The node, m_q, that received a query message checks the grid ID list in the query message. If the grid ID list includes its own grid identifier, m_q stores the source node ID as its parent node, and data ID list, and transmits the query message. Here, the data identifiers that m_p has are added to the data ID list in the query message except for same data identifiers. If the grid ID list includes one grid ID (this means the query issuer is only searching its own grid) and the size of the grid is R, m_p does not need to transmit the query message, so it immediately starts to send a reply message to the query issuer. However, if the grid ID list does not contain its own grid identifier, m_q ignores this message.
- When m_q receives a query message again, it stores the source node ID as its child node. Moreover, it stores the data ID list in the query message when the hop count in the query message is smaller than itself or the hop count is same but the own node identifier is smaller than that of the source node. By this process, the number of replied data items can be reduced.
- Each node, m_r, which has no child node, starts transmitting a reply message to its parent node. The reply message includes the query issuer ID, the query ID, the source node ID, and the reply list. The reply list includes data items with the k highest score that are not included in the data ID list received during query transmission.
- The node, m_t, that received the reply message adds the data items to the reply list except for same data items. When m_t receives a reply message from all its children nodes or a certain time has elapsed since it received a query message, it sends its parent node a reply message.
- When the query issuer, m_p, receives reply messages from all of its children or the nodes in its own grid where the search range is only in its own grid, the top-k query processing is over.

In GridTop, a query message is only sent to the necessary nodes, and this prevents the same data items from being replied. As a result, it achieves small overhead and delay.

VII. Simulations

In this section, we show results of simulation experiments regarding performance evaluation of our proposed method. For the simulation experiments, we used the network simulator, QualNet5.2 [16].

A. Simulation setting

Mobile nodes exist in an area 1000 [m] $\times$ 1000 [m]. The initial position of each node is determined randomly. The number of mobile nodes in the entire network is M. These nodes are stationary in this setting. Each mobile node transmits messages (and data items) using an IEEE 802.11b device whose data transmission rate is 11 [Mbps]. The transmission power of each mobile node is determined so that the radio communication range becomes about 100 [m]. Packet losses and delays occur due to radio interference.

Each mobile node holds 10 data items and it deletes and generate data items at the random time whose range is [30, 3600]. The size of data items is 32 [Byte]. The score of each data item is randomly determined within the range between 0 and 100. Each mobile node can replicate B data items on its storage.

Every mobile node that belongs to a grid issues a top-k query where $k(k_{max} = 100)$ is set as 25, 50, 75 and 100. k_1 in the equation (10) is 50 because the probability that each node designates less than 50 (i.e., 25 and 50) is 50%. We assume that top-k queries are processed by using our proposed method.

In a simulation experiment, the number of buffer size, B, is basically 30 and is varied from 5 to 100 in subsection VII-B. The number of nodes is basically 500 and is varied from 250 to 1000 in subsection VII-C.

In the above simulation model, we randomly determine the initial position of nodes and a node collects the position of nodes and $k + margin$ highest scores, and it then inform that information to all of nodes. After this procedure, the query issuer which is randomly chosen among all nodes issues top-k query every 30 seconds. We evaluate the following criteria during 10800 [sec] at every k.

- *Accuracy of query result:* is defined as the ratio of the numbers of data items acquired by the query issuer that are included in the top-k result to the number of requested data items, k.
- *Query overhead:* is defined as the average total volume of query and reply messages during one query processing.
- *Delay:* is defined as the elapsed time after the query issuer issued a top-k query until it acquired the result.
- *Reallocation overhead:* is defined as the total volume of a message for reallocation during the one query (30 [sec]).

B. Impact of buffer size

We examine the effects of the maximum number of replicas allocated on each node, B. Figure 2 shows the simulation result. In these graphs, the horizontal axis indicates B, and the vertical axes indicate the query overhead in the case of (a), the accuracy of query result in the case of (b), the delay in the case of (c), and the reallocation overhead in the case of (d). In these graphs, "$k = x$" denotes the result that the query issuer designates x as k, "Movement" denotes the overhead related to the movement of nodes, and "Data event" denotes the overhead related to the deletion and generation of data items.

From Figure 2(a), when the buffer size is very small ($B = 5$) in the case that k is large ($k = 100$ and $k = 75$), the overhead is significantly large. This is because the grid size is large, and a query message should be sent to nodes in nearby grids. However, in the case that k is small, the query issuer sends a query message in only its own grid, so the overhead is small. When $B = 10$, the overhead becomes smaller because the grid size becomes small (i.e., the search area becomes smaller). As B becomes large, the overhead slightly increases because the number of replied data items increases, and then the overhead significantly decreases because the search range becomes small. When $B = 60$, as the size of the grid becomes small, the overhead decreases.

From Figure 2(b), when the buffer size is very small ($B = 5$), the accuracy of query result is small due to packet losses. This is because the search range is large, so the packet loss occurs. As B increases, the accuracy of query result increases. It has two reasons. First, the query issuer has more data items with k highest score as its replicas. Second, the replied data items decrease from far nodes because the search range becomes smaller and each node attaches the information on data items to the data ID list in a query message. When $B = 60$, the accuracy of query result decreases in the cases that $k = 25$ and $k = 60$. This is because the size of the grid becomes small, and thus, the packet collisions occur because nodes concurrently send back a reply message.

From Figure 2(c), the delay is small in the case that k is small. This means there are the necessary data items in own grids. When $B = 10$ and $B = 60$, the delay significantly decreases. This is because the size of the grid becomes small.

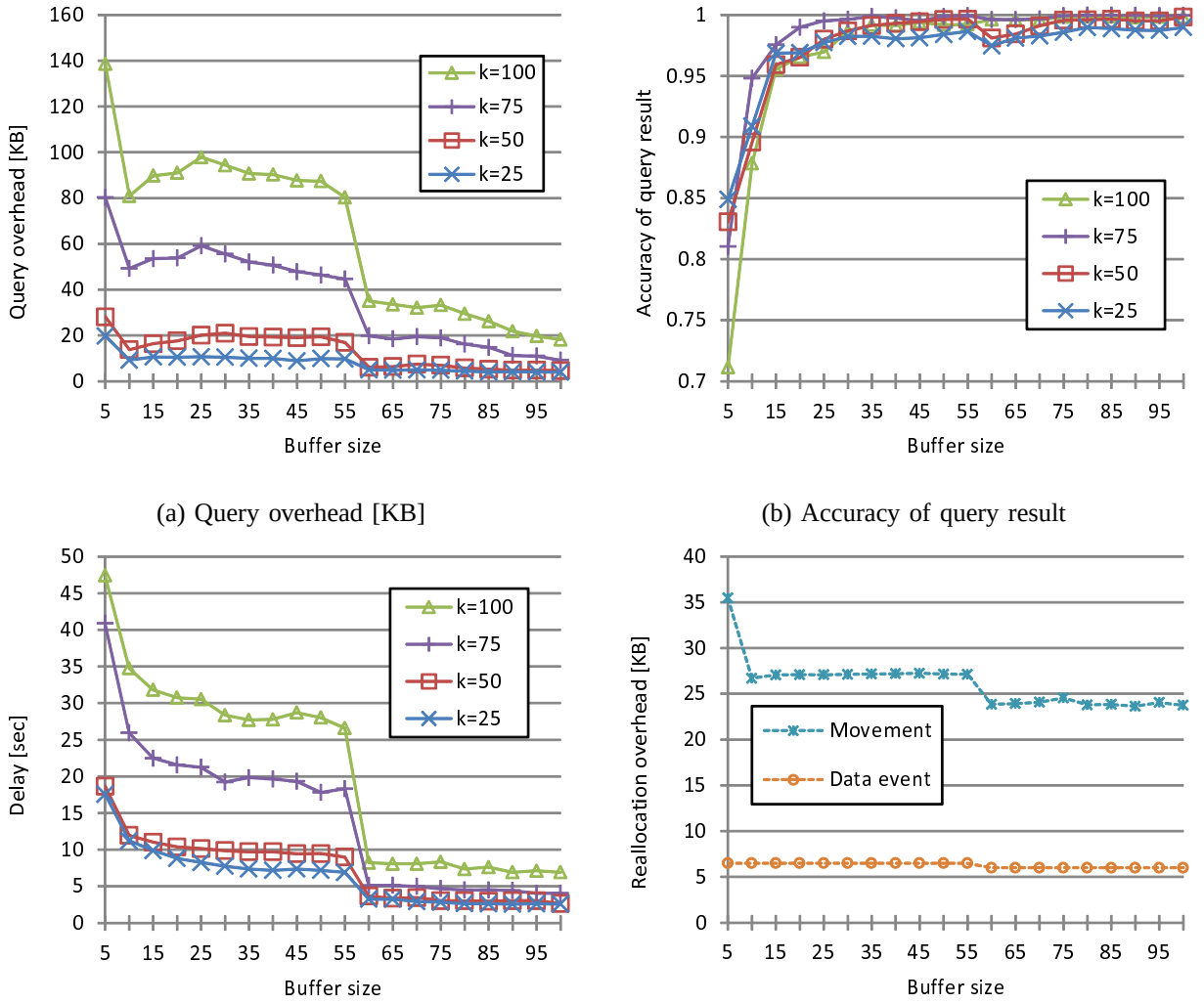
From Figure 2(d), as the buffer size increases, the overhead related to the movement of nodes decreases. This is because as the grid size becomes smaller, the number of messages is smaller. When the grid size is large, the message is flooded in the grid, and thus, the overhead is large. The overhead related to the deletion and generation of data items is almost constant. This is because this overhead is not related buffer size, and related to the number of nodes, k_{max}, the frequency of deletion and generation of data items.

C. Impact of number of nodes

We examine the effects of the number of nodes, M. Figure 3 shows the simulation result. In these graphs, the horizontal axis indicates M, and the vertical axes indicate the query overhead in the case of (a), the accuracy of query result in the case of (b), the delay in the case of (c), and the reallocation overhead in the case of (d).

From Figure 3(a), as the number of nodes increases, the overhead decreases except for the case that $k = 100$ because the search range becomes smaller. When the number of nodes further increase, the number of nodes that transmits the message and replied data items increase, so the overhead increases. In the case that $k = 100$, the overhead constantly increases. This is because the search range gets smaller as the number of nodes increases, but the amount of replied data items increases. As a result, the overhead increases.

From Figure 3(b), the accuracy of query result is high except for the case that the number of nodes is 250. When



(a) Query overhead [KB]

(b) Accuracy of query result

(c) Delay

(d) Reallocation overhead [KB]

Figure 2. The effect of buffer size

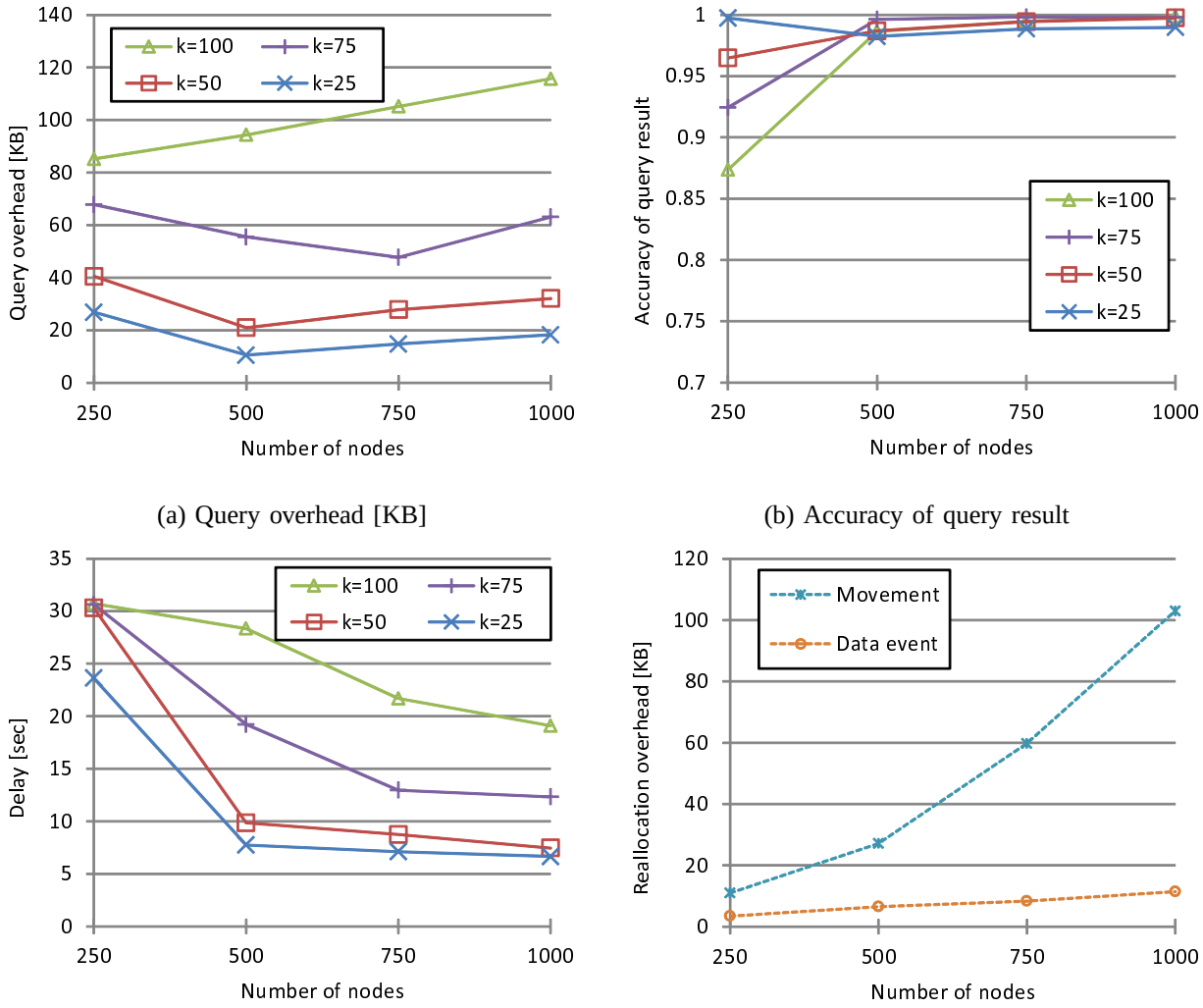
the number of nodes is 250, the number of nodes is too small, and thus, the grid size is very large. Therefore, the packet losses frequent occur, and the data items are not replied back.

From Figure 3(c), as the number of nodes increases, the search range becomes smaller, and thus, the delay decreases. Moreover, the number of replied data items from far nodes, so the transmission delay also decreases.

From Figure 3(d), the overheads related to movement of nodes and deletion and generation of data items increase as the number of nodes increases. This is because the reallocation frequent occurs when the number of nodes is large.

VIII. Conclusion

In this paper, we propose a method which combines top-k query processing and location-based replication in MANETs called $GridTop$ (grid-based top-k query with location-based replication) to achieve small overhead and delay. In this method, the entire area is divided into grids based on the buffer size and the communication range, and the data items are then allocated to nodes in the grids. Since the data items are geographically and uniformly allocated, a query issuer can identify the grids where nodes have the necessary data items. In GridTop, the query issuer only sends a query message to necessary grids to obtain the top-k result. When each node has replicas, the same data items probably are sent back to the query issuer. To prevent it, each node attaches the data identifiers whose data items have the k highest score to a query message. As a result, GridTop achieves the small



(a) Query overhead [KB]

(b) Accuracy of query result

(c) Delay

(d) Reallocation overhead [KB]

Figure 3. The effect of number of nodes

overhead and delay, and also keeps high accuracy of query result.

The simulation result shows that GridTop achieves high performance, where it can acquire the top-k result from nearby grids.

In our proposed location-based replication, we adopted a centralized approach in allocating data items. In our future work, we plan to address a distributed approach.

REFERENCES

[1] R. Akbarinia, E. Pacitti, and P. Valduriez, "Reducing network traffic in unstructured P2P systems using top-k queries," *Distributed and Parallel Databases*, Vol19, No(2–3), pp.67-86, 2006.

[2] W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden, "Progressive distributed top-k retrieval in peer-to-peer networks," In Proc. *ICDE'05*, pp.174-185, 2005.

[3] X. Fan, J. Cao, and W. Wu, "Contention-aware data caching in wireless multi-hop ad hoc networks," *J. Parallel and Distrib. Comput.*, 71, pp.603–614, 2011.

[4] M. Fiore, C. Casetti, and C.F. Chiasserini, "Caching strategies based on information density estimation in wireless ad hoc networks," in *IEEE Tran. on vehicular techinology*, 60(5): pp. 2194–2208, 2011

[5] M. Fiore, F. Minini, C. Casetti, and C.F. Chiasserini, "To cache or not to cache?," in *Proc. Infocom 2009*, pp. 1109–1117, 2009.

[6] R. Hagihara, M. Shinohara, T. Hara, and S. Nishio, "A message processing method for top-k query for traffic reduction in ad hoc networks," *Proc. MDM 2009*, pp.11-20, 2009.

[7] T. Hara, R. Hagihara, and S. Nishio, "Data replication for top-k query processing in mobile wireless sensor networks," *Proc. SUTC 2010*, pp.115-122, 2010.

[8] T. Hara, “Effective replica allocation in ad hoc networks for improving data accessibility,” in *Proc. Infocom 2001*, pp. 1568–1576, 2001.

[9] S. B. Lee, S.H.Y. Wong, K.W. Lee, and S. Lu, “Content management in a mobile ad hoc network: beyond opportunity strategy”, in *Proc. Infocom 2011*, pp. 261–265, 2011.

[10] B. Tang, H. Gupta, and S. Das, “Benefit-based data caching in ad hoc networks,” in *Proc. Infocom 2006*, pp.208–217, 2006.

[11] L. Yin and G. Cao, “Supporting cooperative caching in ad hoc networks,” in *Proc. Infocom 2004*, pp. 77–89, 2004.

[12] H. Matsunami, T. Terada, and S. Nishio, “A Query Processing Mechanism for Top-k Query in P2P networks,” *Proc. SWOD*, pp.84-87, 2005.

[13] N. Padhariva, A. mondal, V. Goval, R. Shankar, and S.-K. Madria, “EcoTop: An Economic Model for Dynamic Processing of Top-k Queries in Mobile-P2P Networks,” *Proc. DASFAA*, pp. 251-265, 2011.

[14] A. Vlachou, C. Doulkeridis, K. Norvag, and M. Vazirgiannis, “On Efficient Top-k Query Processing in Highly Distributed Environments,” *Proc. SIGMOD*, pp.753-764, 2008.

[15] P. Kalnis, W.S. Ng, B.C. Ooi, and K.-L. Tan, “Answering Similarity Queries in Peer-to-Peer Networks,” *Information Systems*, Vol.31, No.1, pp.57-72, 2006.

[16] Scalable Networks: makers of QualNet and EXata, the only multi-core enabled network simulation and emulation software.[Online].Available: http://www.scalable-networks.com/.