International Journal of Computer Applications (0975 – 8887)
Volume 169 – No.2, July 2017

Adaptive Hybrid Methods for Cooperative Caching Consistency in Mobile Ad-Hoc Networks

Ahmed A. A. Gad-Elrab
Department of Mathematics, Faculty of Science
Al-Azhar University, Cairo, Egypt

ABSTRACT

In mobile environment, enhancing data consistency among data caches by mobile clients and data residing in a server is a big problem due to the mobility of nodes. Many updating schemes have been proposed to solve this problem. However, these updating schemes produce a high updating cost which consumes most of the limited resources of mobile clients as battery power. In this paper, to solve this problem, an adaptive hybrid data-based cache consistency scheme is proposed. The proposed scheme classifies the data items into push data items and pull data items. Push data items need to be updated periodically by their owners while pull data items are updated based on the request of their cache nodes. Also, the new scheme proposes two updating methods which are called separate path method and *k*-path tree method. In the first method, the updating mechanism uses separate paths to send update data to cache nodes of a certain data item. While the second method constructs a *k* path tree among cache nodes, then it sends the updating data through this tree, level by level. In addition, the proposed scheme does not only give the ability of sending update data to the owner of data, but also it gives this ability to any cache node that has the data items. Therefore, the proposed scheme can maintain the data consistency, decrease unnecessary communication overhead, and reduce access latency. The results of conducted simulations have shown that the proposed consistency scheme is much better than existing methods.

General Terms

Data management, Mobile Ad-Hoc

Keywords

MANETs, caching, cache management, communication overhead, cache consistency.

1. INTRODUCTION

Recently, mobile ad hoc networks (MANETs) have a great attention from researchers due to their importance in many applications and situations as data gathering, disaster scenarios, and data dissemination. MANETs are infrastructure-less wireless networks that depend on multiple hops for communication and they use access points (APs) only for connecting to outside distributed networks as the Internet. MANETs are different from Infrastructure-dependent networks which are confined with single-hop wireless communication and rely on base stations or APs to forward messages that are sent or received by the mobile nodes.

All mobile devices in MANETs are distributed over an area in which access to external data is achieved by being directly connected to APs through one or more mobile nodes. These nodes known as router nodes to other mobile nodes for accessing the APs. In MANETs, the query delay for receiving responses to queries may be is very high, if there is a large number of mobile nodes that access APs which produces a heavy traffic in the network. In addition, MANETs have less stable wireless links and high bandwidth utilization to deal with its dynamic topology. Also, the energy utilization and mobility of users are the most important challenges in deciding the access latency and network overhead. Thus, in such dynamic networks, the cooperation among all mobile nodes is very important to perform their tasks in efficient and effective ways.

The biggest problem in MANETs is data query management which aims to find an efficient policy for responding to each data request initiated by mobile nodes such that the energy consumption and the query delay are minimized. The most efficient solution to this problem is data caching. Data caching is a policy to store and distribute data items at different mobile nodes in the networks to improve the overall network performance by minimizing heavy traffic occurring on the data owner and reducing the access latency that are existed by user queries. These nodes are called cache nodes. In addition, the cached data may be shared among multiple mobile users through the network. Hence, cooperative caching is needed for minimizing the load on the data owner which replies alone for all possible user requests. Moreover, the query delay can be minimized by eliminating traversing multiple hops to reach the data owner and it can be directed to any nearby cache node that has a data item.

The information retrieval process in such dynamic network is very difficult. In addition, the cached information in any cache node must be identical to the same information present in the data owner. This identicalness represents the consistency degree of data items in the network. Therefore, maintaining the consistency degree among all cache nodes for any data item is difficult to be guaranteed. Thus, effective and efficient data consistency methods are needed to improve the data consistency in such mobile networks by considering user requirements, energy consumption, query delay, and network traffic.

In this paper, new cache consistency scheme called an adaptive push-pull data-based cache consistency scheme is proposed. The proposed scheme classifies the data items into push data items and pull data items based on their owner's decision. This classification is known as data level classification because it depends on the data itself while some of existing schemes depend on a node level classification which classifies the mobile nodes into push and pull node. In the proposed scheme, push data items need to be updated randomly or periodically by their owners while pull data items are updated based on the request of their cache nodes. Also, the new scheme proposes two updating methods which are called separate path method and k-path tree method. In the first method, the updating mechanism uses separate paths for sending update data to all cache nodes of a certain data item. While the second method constructs a k paths tree among all cache nodes, then it sends the updating data through this tree, level by level. In addition, the proposed scheme does not only give the ability of sending update data to the owner of data, but also it gives this ability to any cache node that has the data items. With this authority, the proposed scheme can maintain the data consistency, decrease unnecessary communication overhead, and reduce access latency.

2. RELATED WORK

A lot of caching schemes have been proposed to improve the information retrieval, deliver higher data availability [1], [2], [3] and increase the data consistency [4], [5], [6] in mobile environments. There are three main categories of cache consistency schemes: push-based schemes which rely on the data owner or the server for cache updates, pull-based schemes which are client-based and attain cache updates on requests to the data owner or the server, and hybrid schemes which use push and pull policies based on a client decision where a client can select to receive the updating periodically from the server as push-based scheme or when the client ask the server for new update as pull-based scheme. Many of the cache consistency schemes either adopt push-based or pull-based techniques. For instance, [5] and [7] rely on pull-based cache-initiated consistency management schemes while [4] and [8] use push-based server update mechanisms.

In push-based schemes, the owner of data or server uses the update activities of each data item to send a report to its cache nodes. Yin et al. [6] proposed a generalized cache replacement policy for mobile environment called IR method. In IR method, the server creates an IR entry list which carries the updated IDs of data items and the time stamps of the updated history. Then the server broadcasts these IRs periodically in the network. If a client node wants to generate a new query for requesting a certain data item, first it waits for the periodic IR to validate its cache. If its cache is valid, then the query is transmitted. If it is invalid or modified, it usually waits for the next periodic IR. Also, Cao et al. [9] proposed an IR-based algorithm to reduce network traffic. Krishnamurthy et al. [4] proposed a piggyback invalidation scheme called PSI mechanism to maintain a cache coherency in Web proxy caches to reduce overall costs. In PSI mechanism, servers are piggybacking the list of resources that have changed since the last access by a client on a reply to the proxy client. Then, the proxy client invalidates cache entries on the list and it can extend the lifetime of entries not on the list.

In pull-based schemes, the updating process is based on the requests that issued by cache nodes. There are two common categories of pull-based algorithms, time to live (TTL) algorithms and client validation algorithms. In TTL algorithms, each cache node stores a TTL value which is assigned by the server in its cache. Several TTL-based algorithms have been proposed. In [10], Cao et al. proposed an adaptive TTL-based cache consistency scheme. In [11], an adaptive algorithm was discussed to maintain consistency within a client server-based mobile network. Tang et al. [12] described a TTL-based consistency scheme for unstructured peer-to-peer networks which have millions of nodes that share data through searching and replication. However, replication can improve the data sharing but in turn it complicates data consistency. In [13], Jung et al. proposed a fixed TTL algorithm.

While in client validation algorithms, a client sends validation requests to the server based on certain criteria, then the server use a validation process. In this validation process, if the data are not an identical copy of the same data on the server, then the server responds with an invalid data message or a valid data message. An invalid message means that the data were modified and changed while a valid message means that the data were not modified or changed. It is very much like piggyback cache validation [5]. In [14], an invalidation report (IR)-based cache invalidation algorithm, which can significantly reduce the query latency and efficiently utilize the broadcast bandwidth, was proposed. The IR-based cache invalidation leads to two major drawbacks. First, when the server updates a hot data item, all clients must query the server and get the data from the server separately, which wastes a large amount of bandwidth. Second, there is a long query latency associated with this solution since a client cannot answer the query until the next IR interval.

In hybrid schemes, a client decides to receive periodically updates from the server as push policy or receive an update data when a client requests new update from the server as pull policy. In [15], Selvin et al. proposed push-pull cache consistency mechanism for cooperative caching in mobile ad hoc environments called 2P2C mechanism. In 2P2C, client nodes are classified into push and pull nodes as a node level classification by using a registration process. This registration process is performed by each client in the network. In this registration process, each client decides to be a push or pull client. Based on this registration process, 2P2C mechanism follows both a server-based consistency scheme and a client-based consistency scheme. In addition, both the server and the cache nodes can accept client registrations messages. However, 2P2C may not achieve a high consistency degree, because the pull clients may have non-identical copies of the same data item.

Most of existing schemes do not meet all required issues for information retrieval and data consistency as minimizing communication overhead, minimizing network traffic, minimizing energy consumption, and maximizing the consistency degree of data items in the networks.

3. CACHE CONSISTENCY PROBLEM IN MOBILE AD-HOC NETWORKS

The cache consistency problem in MANETs, CCPM, is how to keep all copies of a certain data item in the system with last update content by considering the consumed energy for updating operation, the update time delay, and the mobility nature of MANETs. In the rest of this section, the assumptions and models will be introduced then the CCPM will be formulated.

3.1 Assumptions and Models

Mobile ad-hoc networks consists of a set of mobile nodes, $MN = \{mn_1, mn_2, \ldots, mn_i, \ldots, mn_K\}$ and each mobile node, mn_i, has a limited cache C_i and a limited battery, E_i. A set of data items, $DI = \{d_1, d_2, \ldots, d_j, \ldots, d_R\}$ where each data item d_j can be distributed and cached at different mobiles nodes in the networks based on the issued requests by the mobile nodes and the available size of their cache memory. The set of copies of each data item d_j is denoted as $RD_j = \{rd_{j,1}, rd_{j,2}, \ldots, rd_{j,x}, \ldots, rd_{j,Xj}\}$. We assume that each mobile node can request a certain data item from its owner or from any nearby cache node that has a copy of this data item. We assume that the owner of each data d_j can send an update for all mobile nodes in the system that have a copy of this data items. Any mobile node that has a copy of any data item is called a *cache node*, cn_i. We assume that each update operation for any data item in the system is numbered by an integer value, un_j in ascending order as 1, 2, 3, ..., and so on. We assume that each cache node has a list of copies of some of data items which is denoted as CD_i. We assume that each mobile node that receives an update operation from the owner or from any other cache node for updating a certain data item d_j in its cache, will store the data item id, the number of last update number of this data item, lu_j, and the time of received update, tru_j in an array structure called *Stored Update Copies*, SUC as shown in Fig.1. We assume that any cache node in the system has an array structure called *Cache Nodes Info*, CNI, to store all information about all nodes that receive a copy of a certain data item from this cache node as node id, *Nid* and list of data item copies with its sending time pairs, ($rd_{j,x}$, $st_{j,x}$). CNI is shown in Fig.2. In any update operation of a data item d_j, the owner will send an update message that contains the new content of data item and the number of current update, un_j. Then the cache node that receives this message, will update its SUC and resend the same message to its related cache nodes that exist in its CNI. The total cost of each update operation is based on the consumed energy, $uCE(d_j, un_j)$, and the update time delay, $uTD(d_j, un_j)$, which represents the total time interval for arriving the update message to each cache node. $uCE(d_j, un_j)$ and $uTD(d_j, un_j)$ are defined as follows.

Data Item ID, $rd_{j,X}$	Last Update Number, lu_j	Time of Received Update, tru_j

Fig. 1: SUC, Structure of Stored Update Copies

Node ID, *Nid*	list of data item copy and its sending time pairs ($rd_{j,x}$, $st_{j,x}$)

Fig. 2: CNI, Structure of Cache Node Info.

$$uCE\left(d_j, un_j\right) = \sum_{p=1}^{X_j} uce\left(rd_{j,p}, un_j\right) \qquad (1)$$

$$uTD\left(d_j, un_j\right) = \sum_{p=1}^{X_j} utd\left(rd_{j,p}, un_j\right) \qquad (2)$$

where $uce(rd_{j,p}, un_j)$ and $utd(rd_{j,p}, un_j)$ are the consumed energy and the total time delay for updating all copies of data item d_j, by the operation number un_j, respectively.
By using Equations 1 and 2, for a data item d_j, the total cost of its update operation is defined as follows.

$$uCost\left(d_j, un_j\right) = w_1 * uCE\left(d_j, un_j\right) + w_2 * uTD\left(d_j, un_j\right) \qquad (3)$$

where w_1 and w_2 are weight values that represent the importance degree of the consumed energy and the delay time for the system, such that $w_1 + w_2 = 1$. By using Equation 3, the total cost for all data items is defined as follows.

$$uTC(DI) = \sum_{j=1}^{R} \sum_{un_j=1}^{H_j} uCost\left(d_j, un_j\right) \qquad (4)$$

where H_j is the total number of update operations for data item d_j.
In addition, we assume that the cache consistency can be measured by computing the number of cache nodes that have the last update number of a data item d_j which can be found in SUC of each cache node. Here, we define the consistency as a ratio of number of cache nodes that have the last update number to the total number of cache nodes of a data item d_j. This ratio is called *consistency degree*, cDeg(d_j) and is defined as follows.

$$cDeg\left(d_j\right) = \frac{\left|MCN\left(d_j\right)\right|}{\left|CN\left(d_j\right)\right|} \qquad (5)$$

Where $MCN(d_j)$ and $CN(d_j)$ are the set of cache nodes that have the last copy of data item d_j and the set of all cache nodes that have a copy of data item d_j, respectively. By using Equation 5, the final consistency degree for all data items in DI is equal to the average of all consistency degrees of all data items and is defined as follows.

$$cFDeg(DI) = \frac{\sum_{j=1}^{R} cDeg\left(d_j\right)}{R} \qquad (6)$$

3.2 Problem Formulation

The main objective of CCPM is maximizing the consistency degree and minimizing the total cost of updating operations for each data item. So, CCPM is an optimization problem and is formulated as follows.

$$\boldsymbol{Maximize} \qquad cFDeg(DI)$$

such that

$$uCE(d_j, un_j) \le CET_{d_j} \quad \forall\, d_j \in DI \qquad (8)$$

$$uTD\left(d_j, un_j\right) \le TDT_{d_j} \quad \forall\, d_j \in DI \qquad (9)$$

$$\sum_{z=1}^{|CD_i|} Size(cd_z) \le Size(C_i) \forall\, mn_i \in MN \qquad (10)$$

$$\sum_{q=1}^{Q} \left(En_{q,send} + En_{q,recieve}\right) \le E_i \quad \forall\, mn_i \in MN \qquad (11)$$

constraint 8 means that the consumed energy for updating copies of each data item d_j must be less than or equal to a predefined threshold for the consumed energy called CET_{dj}. Constraint 9 means that the update time delay for updating copies of each data item d_j must be less than or equal to a predefined threshold for the time delay called TDT_{dj}. Constraint 10 means that the total size of the cached data items in each node must be less than or equal to the available size of its cache. Constraint 11 means that the total consumed energy by a node for sending and receiving update messages is less than or equal to the available energy of its battery.

4. THE PROPOSED ADAPTIVE CACHE CONSISTENCY METHODS

In this section, to solve the CCPM problem in MANETs, a new updating scheme called an *Adaptive Push-Pull Data-Based for Caching Consistency, A2PD2C* is proposed. The basic idea of A2PD2C is based on (1) *Data classification*: which means that classifying the set of data items into push data items and pull data items. In other words, the determination of push or pull data depends on the data level and it did not depend on the node level as in 2P2C scheme [15]. (2) *Data updating*: which means that each owner of data items will send periodically update messages for updating all push data items that are cached on other mobile nodes. Whereas for any pull data item, the owner will send the update message to all mobile nodes which has cached copies of this pull data item if a certain mobile node asked for the updating version.

Based on this basic idea, A2PD2C can keep all data items with high consistency level and it can minimize the overhead of the updating process which produces a large amount of sending and receiving messages.

4.1 The Proposed Scheme

The proposed scheme A2PD2C consists of three modules: *Marking module*, *updating module*, and *replacement module*. These three modules are described as follows.

4.1.1 Marking module

In this module of A2PD2C, each owner of data items marks each data item with a bit flag called *Update Type Flag* (UTF). The value 0 of UTF means that this data item is pull data item while the value 1 means that this data item is push data item. By using this UTF value, each cache node that has a set of cached data items can classify their cached data items into push data items (all cached items with UTF =1) and pull data items (all cache items with UTF=0).

4.1.2 Updating module

In this module of A2PD2C, each owner of data items sends an update message for each push data item periodically by determining the best value of updating interval for sending this update message. While for pull data item, the owner node will send an update message if at least one of cache node for this data item asked for updating the data. Here, the problem is how the new update message for a certain data item can be sent to all cache nodes such that the total cost of updating process is minimized. In other words, how to minimize the number of traversing messages for sending a new update data to all cache nodes. To achieve this goal, A2PD2C proposes two different methods for sending any new update data message. These two methods are described as follows.

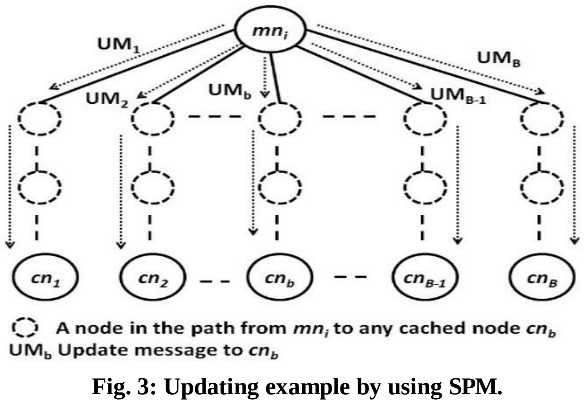
mn_i
UM_1
UM_2
UM_b
UM_{B-1}
UM_B
cn_1
cn_2
cn_b
cn_{B-1}
cn_B
A node in the path from mn_i to any cached node cn_b
UM_b Update message to cn_b

Fig. 3: Updating example by using SPM.

The first method is called *Separate Path Method* (SPM). In this method, A2PD2C sends the new updating data along a separate path for each cache node. For example, if a node mn_i has a data item d_j, and there are B cache nodes that have a copy of this data item. If mn_i wants to send a new updating data to these B nodes, it will send this new message along B routing paths, separately, as shown in Fig.3. This SPM will be used by each owner node and each cache node to process the updating data message.

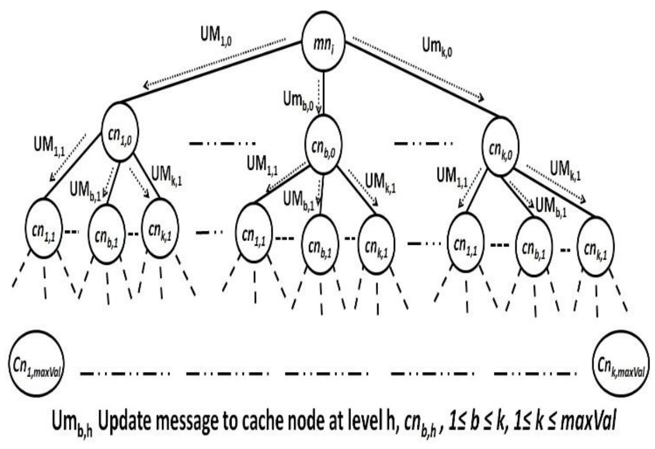

Fig. 4: Updating example by using *k*PTM.

The second method is called *k-Path Tree Method* (*k*PTM). In this method, A2PD2C sends the new updating data by using a tree structure with *k* roots as follows.

1. Assume that there is H cache nodes in CNI list of the owner of data (cache node) of data item d_j.
2. *k*PTM selects k nodes of CNI list, where $k \geq 2$.
3. For the reminder number of (H - k) nodes in CNI list, *k*PTM constructs k groups of nodes. Each group of nodes is denoted as $g(cn_s, d_j, l)$ where $1 \le s \le k$ and l represents the tree level in constructed tree for *k*PTM. The value of l is started with 0 (level 0, the first level at owner node), and is ended with $maxLV_k$ (last level of the constructed tree). The value of $maxLV_k$ depends on the number of cache nodes in CNI list, H. Here, each $g(cn_s, d_j, l)$ can contain some elements of CNI or is empty group such that

$$CNI = \bigcup_{s=1}^{k} g(cn_s, d_j, l) \qquad (12)$$

4. *k*PTM sends the new updating data to each node of the k selected nodes and associates each message with one of the constructed groups.
5. Each selected node of k nodes that receives the updating data will select new k nodes from the received group of nodes, $g(cn_s, d_j, l)$ and repeat steps 3 and 4.
6. *k*PTM will repeat this process until all cache nodes receive the updating data. The last node receives this updating message represents the $maxLV_k$ level of the constructed updating tree, UT_{kPTM}. Fig.4. shows an example of *k*PTM method.

4.1.3 Replacement module

Due to the limited size of the cache in each mobile node in MANETs, the most important issue is not only the cache update, but also the cache replacement, which means which data item in the cache must be replaced by a new received data item. There are a lot of replacement schemes that have been proposed in the literature [16], [17], [18], [19], [20], [21]. In [21], Ahmed et al. proposed a new replacement scheme called PMCR for mobile environments. PMCR selects the data items for replacing from the cache based on Markov Model, predicted region, and cost function. To determine the radius of a predicted region, PMCR uses the root-mean squared distance. The replacement scheme is out of scope of this paper, so A2PD2C uses PMCR as a cache replacement strategy. The three modules architecture of A2PD2C is shown in Fig.5.

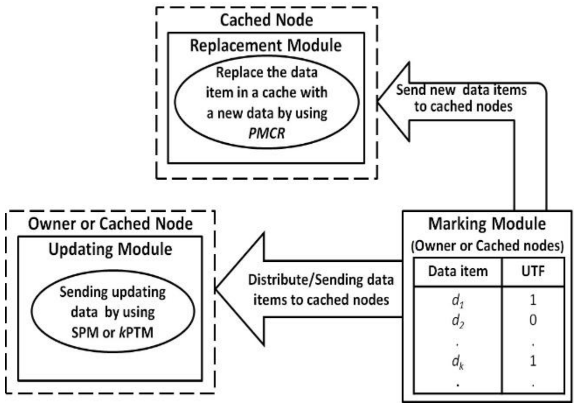
Cached Node
Replacement Module
Replace the data item in a cache with a new data by using PMCR
Send new data items to cached nodes
Owner or Cached Node
Updating Module
Sending updating data by using SPM or kPTM
Distribute/Sending data items to cached nodes
Marking Module (Owner or Cached nodes)
Data item
UTF
d1 1
d2 0
dk 1

Fig. 5: Three modules architecture of A2PD2C.

4.2 Updating Cost and Time Complexity of A2PD2C

To show the efficiency of the proposed scheme A2PD2C, in this section the time complexity of A2PD2C will be determined for the two-proposed updating methods SPM and *k*PTM. Also, the updating cost of two methods which is based on the number of sending and receiving updating messages along all paths from the owner of the data item to the cache nodes. Assume that the current number of cache nodes for data item d_j which is owned by mn_i is B nodes. Assume that the path length from mn_i to any cache node is $PL(mn_i, cn_b)$, $1 \le b \le B$.

The updating cost of SPM and *k*PTM depends on the path length of routing paths from the owner of the data item to the cache nodes. But the two methods have different values which can be calculated as follows.

For SPM, the updating cost, $UC_{SPM}(mn_i, d_j)$, is equal to the sum of all updating costs along the separate paths to the cache nodes and is determined by the following equation.

$$UC_{SPM}(mn_i, d_j) = \sum_{b=1}^{k} PL(mn_i, cn_b) \quad (13)$$

While for *k*PTM, the updating cost, $UC_{kPTM}(mn_i, d_j)$, is equal to the sum of all updating costs along all k tree branches to all cache nodes and depends on the maximum number of levels, $maxLV_k$, in the updating tree. This cost is determined by the following equation.

$$UC_{kPTM}(mn_i, d_j) = \sum_{l=1}^{maxLV_k} \sum_{s=1}^{k} \sum_{p=1}^{k} PL(cn_{s,l-1}, cn_{p,l}) + \sum_{s=1}^{k} PL(mn_i, cn_{s,0}) \quad (14)$$

The time complexity of SPM, TC_{SPM}, is based on the path length of routing paths from the owner of the data item to the cache nodes. Thus, the time complexity of SPM method is defined as follows.

$$TC_{SPM}(mn_i, d_j) = O(Max_{1 \le b \le B}\{PL(mn_i, cn_b)\}) \quad (15)$$

While the time complexity of *k*PTM, TC_{kPTM} , is based on the $maxLV_k$ value of the last level in the constructed tree. Thus, the time complexity of *k*PTM method is defined as follows.

$$TC_{kPTM} = O(maxLV_k) \quad (16)$$

Here, the value of $maxLV_k$ depends on the number of cache nodes of a data item d_j which is B. So, the determination of $maxLV_k$ value is demanded for getting the values of UC_{kPTM} and TC_{kPTM}. To determine this value, the following theorems and their proofs are introduced as follows.

Theorem 1: *For kPTM method, if the number of cache nodes is B and the dividing value is k for any data item d_j, then the number of reminder nodes, NRN_i, for each branch in a level i of its updating tree, UT_{kPTM} is determined by the following equation.*

$$NRN_i = \frac{B(k-1) - k^{l+1} + k}{k^l(k-1)} \quad (17)$$

Proof: Here, **Theorem 1** will be proofed by using the mathematical induction based on the level value l as follows.

Step 1: for $l = 1$ (level 1), the number of reminder nodes after selecting k nodes is (B-k) nodes and after constructing k groups of the reminder nodes, then the number of reminder nodes for each branch in this level is determined as follows.

$$NRN_1 = \frac{B-k}{k} \quad (18)$$

multiply the right-hand side of equation 18 by $(k-1)/(k-1)$, we get

$$NRN_1 = \frac{B-k}{k} \cdot \frac{k-1}{k-1} = \frac{B(k-1) - k^2 + k}{k(k-1)} \quad (19)$$

thus, equation 17 is true at $l = 1$.

Step 2: for $l = q$ (level q), assume that equation 17 is true and

$$NRN_q = \frac{B(k-1) - k^{q+1} + k}{k^q(k-1)} \quad (20)$$

Step 3: for $l = q+1$ (level q+1), the number of nodes in the received set of cache nodes is NRN_q at any node in level q+1. By applying *k*PTM at level q+1, the number of reminder nodes is determined as follows.

$$NRN_{q+1} = \frac{NRN_q - k}{k} \quad (21)$$

by substituting from equation 20 in equation 21, we get

$$NRN_{q+1} = \frac{\frac{B(k-1) - k^{q+1} + k}{k^q(k-1)} - k}{k} \quad (22)$$

$$NRN_{q+1} = \frac{B(k-1) - k^{q+1} + k - k.k^q(k-1)}{k^q(k-1).k} \quad (23)$$

$$NRN_{q+1} = \frac{B(k-1) - k^{q+1} + k - k^{q+2} + k^{q+1}}{k^{q+1}(k-1)} \quad (24)$$

$$NRN_{q+1} = \frac{B(k-1) - k^{(q+1)+1} + k}{k^{q+1}(k-1)} \quad (25)$$

from equation 25, we get that equation 17 is true at $i = q+1$. Thus, theorem 1 is proofed.

Theorem 2: *For kPTM method, if the number of cache nodes is B and the dividing value is k for any data item d_j, then the maximum number of levels, $maxLV_k$, of its updating tree, UT_{kPTM}, is determined by the following equation.*

$$maxLV_k = Log_k[B(k-1)+k] - 1 \quad (26)$$

Proof: by using **Theorem 1,** the number of reminder nodes for each branch in any level *l* is defined as follows.

$$NRN_l = \frac{B(k-1) - k^{l+1} + k}{k^l(k-1)} \quad (27)$$

assume that level *l* is equal to $maxLV_k$, so the number of reminder nodes for each branch at level $maxLV_k$ is equal to zero. Thus

$$NRN_{maxLV_k} = \frac{B(k-1) - k^{maxLV_k+1} + k}{k^{maxLV_k}(k-1)} = 0 \quad (28)$$

$$B(k-1) - k^{maxLV_k+1} + k = 0 \quad (29)$$

$$k^{maxLV_k+1} = B(k-1) + k \quad (30)$$

by taking the *Log* function for base *k*, we get

$$maxLV_k = Log_k[B(k-1)+k] - 1 \quad (31)$$

From equation 31, **Theorem 2** is proofed.

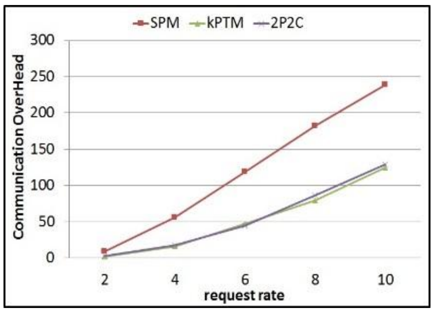

Fig. 6: Comm. overhead vs. request rate

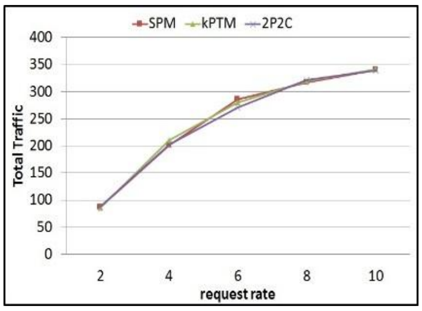

Fig. 7: Total traffic vs. request rate

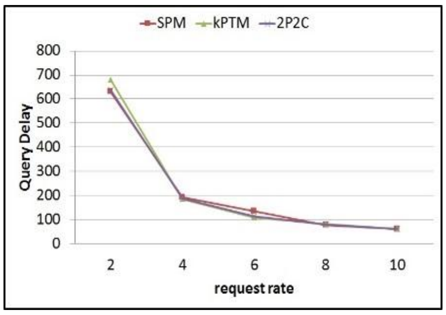

Fig. 8: Query delay vs. request rate

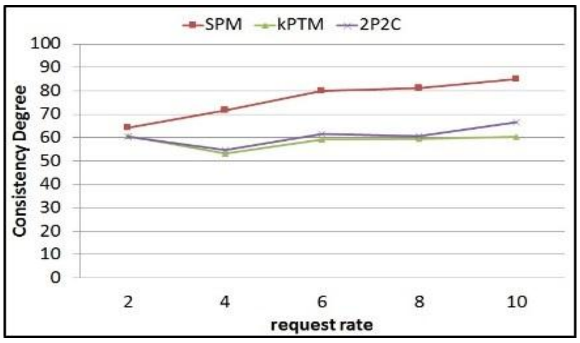

Fig. 9: Consistency degree vs. request rate

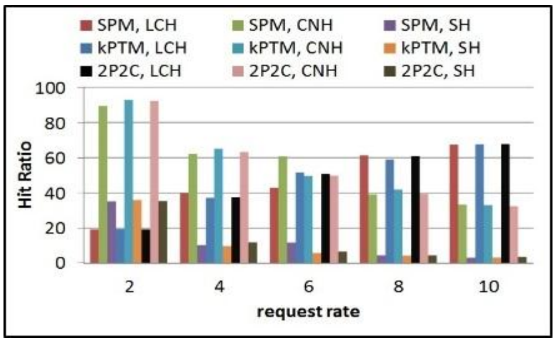

Fig. 10: Hit ratio vs. request rate

5. SIMULATION AND RESULTS

In this section, the evaluation of the proposed scheme A2PD2C is introduced by comparing the two proposed updating methods with 2P2C scheme [15]. OMNet++ Simulator [22] is used to simulate the proposed A2PD2C and 2P2C schemes. OMNet++ is a network simulator that is widely employed to simulate a layered environment in wired or wireless environments. The simulation settings and parameters are summarized in Table 1.

Table 1. Simulation Parameters

Simulation Parameter	Value
Area size	500m x 500m
Mobility model	Random Way Point (RWP)
# of mobile nodes	15 … 75
Speed of a mobile node	1, 2, 3, 4, 5, 6 m/s
Cache size	50…250
Request rate	2, 4, 6, 8, 10
Update rate	2, 4, 6, 8, 10
Split values, *k*	1… 6

Here, mobile nodes are made to move in a 500 m × 500 m area for a simulation time of 900s. A random way point model is adopted as a mobility model for all mobile nodes in the simulation setup. For analysis, traffic, delay, update rate, speed of a node, request rate, and cache size are considered as simulation parameters. Also, each experiment is repeated 5 times and the average was taken. The used performance metrics are described as follows.

- ***Communication overhead:*** This is the overhead that arises due to transmissions of additional messages to maintain consistency between the cache clients and the server. It is represented in terms of number of messages/second.
- ***Traffic:*** This traffic includes requests for an item, forwarding of the requests, and replies for the requests. It is given in terms of number of messages/second.
- ***Query delay***: The delay between the query requests and the time at which the data arrive at the requesting.
- ***Consistency degree:*** This is the ratio of number of cache nodes that have the last update of a certain data item to their total number of cache nodes in the network.
- ***Hit ratio:*** This is used to determine the success rate of requests generated within the network. Here the hit ratio is classified into three classes, namely:
  - ***Local cache hit, LCH:*** This type of hit arises if the requested data are present in the cache of the requestor node itself.
  - ***Cache node hit, CNH:*** If the requests are serviced by

intermediate cache nodes, then it leads to a cache node hit.

- ***Server hit, SH:*** In this case, all the previous attempts end up with a miss, and then the request is forwarded to wards the server, which in turn serves the requestor with the desired data.

5.1 Different request rate

Fig.6, Fig.7, Fig.8, Fig.9, and Fig.10 show the communication overhead, total traffic, query delay, consistency degree, and hit ratio of SPM, *k*PTM, and 2P2C against different values of request rate when the number of data items was 10, the number of nodes was 30 nodes, the update rate was 2, the cache size was 100MB, and the mobile node speed was 1m/s. As shown in Fig.6, the communication overhead increases as request rate increases, this is because the existence of increasing in request rate will increase the number of cache nodes in the network for each data item. Thus, a higher number of communication messages are needed to maintain the consistency degree among all cache nodes. In addition, as shown in Fig.6, the communication overhead of SPM is much larger than *k*PTM and 2P2C methods. This is because SPM uses a normal path from owner of a data item to a cache node while *k*PTM uses a tree structure to send update messages for all cache nodes and 2P2C sends update messages based on normal path but with a node level (push or pull node). Also, as shown in Fig.6, for higher values of the request rate, *k*PTM is better than 2P2C method, this because *k*PTM sends update messages based on data level (push or pull data) while 2P2C uses a node level for maintaining consistency degree among cache nodes. As shown in Fig.7, the total traffic increases as request rate increases, this is because the increasing in request rate will increase the number of cache nodes for each data item and increase the number of forwarding and replying messages in the network. In addition, the total traffic for the three methods is almost the same.

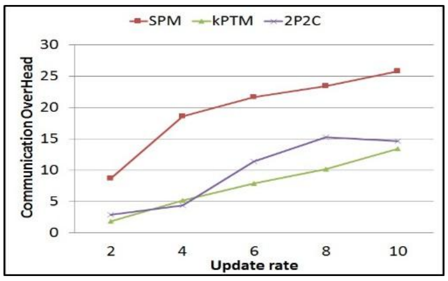

Fig. 11: Comm. overhead vs. update rate

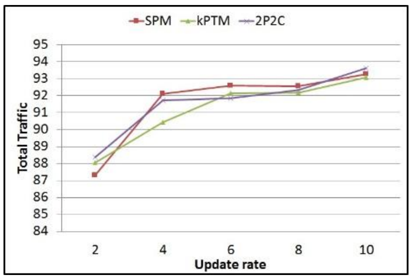

Fig. 12: Total traffic vs. update rate

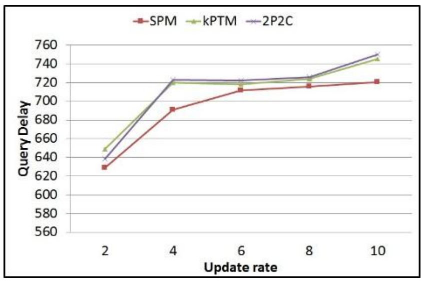

Fig. 13: Query delay vs. update rate

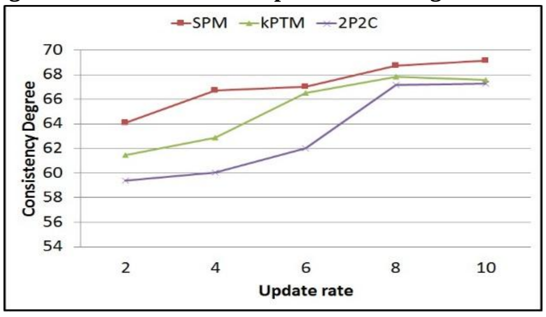

Fig. 14: Consistency degree vs. update rate

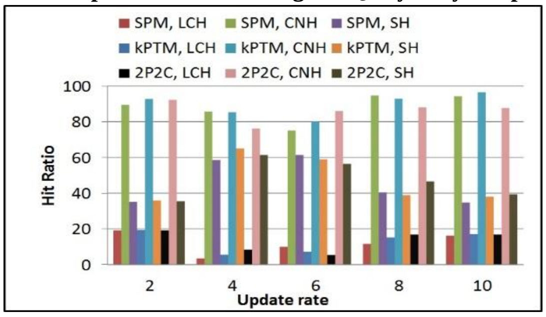

Fig. 15: Hit ratio vs. update rate

As shown in Fig.8, the query delay decreases as request rate increases, this is because the increasing in request rate will increase the number of cache nodes in the network for each data item. Thus, any cache node or a local cache of a node can reply to data request directly without a need to forward the request to the owner node. This replying will minimize the query time delay. In addition, as shown in Fig.8, for low values of request rate, the query delay of SPM and 2P2C is less than the query delay of *k*PTM while for the higher values, the query delay of the three methods is almost the same. As shown in Fig.9, the consistency degree increases as request rate increases, this is because the existence of increasing in request rate will increase the number of cache nodes in the network for each data item. Thus, a higher number of updating messages are needed to maintain the consistency degree among all cache nodes. In addition, as shown in Fig.9, the consistency degree of SPM is much larger than *k*PTM and 2P2C methods. This is because SPM uses a normal path from owner of a data item to each cache node which increases the success possibility to reach the destination cache nodes. While *k*PTM uses a tree structure to send any update message for all cache nodes and 2P2C sends update messages based on normal path but with a node level. Also, as shown in Fig.9, the consistency degree of *k*PTM and 2P2C is almost the same. As shown in Fig.10, the server ratio decreases as request rate increases while the local hit ratio and cache hit ratioincrease as request rate increases. this is because the increasing in request rate will increase the number of cache nodes for each data item and increase the number of data items in a local cache of a node. In addition, the local hit ratio and cache hit ratio of the proposed method SPM and *k*PTM are better than 2P2C method.

5.2 Different update rate

Fig.11, Fig.12, Fig.13, Fig.14, and Fig.15 show the communication overhead, total traffic, query delay, consistency degree, and hit ratio of SPM, *k*PTM, and 2P2C against different values of update rate when the number of data items was 10, the number of nodes was 30 nodes, the request rate was 2, the cache size was 100MB, and the mobile node speed was 1m/s. As shown in Fig.11, the communication overhead increases as update rate increases, this is because as update rate increases a required number of communication messages are needed to maintain the consistency degree among all cache nodes. In addition, the communication overhead of SPM is much larger than *k*PTM and 2P2C methods. This is because using of updating tree and the using of node level concept in *k*PTM and 2P2C, respectively, decreases the required number of communication messages. While SPAM uses a normal path from owner of a data item to a cache node 2 which increases the required number of communication messages. Also, for higher values of an update rate, *k*PTM is better than 2P2C method, this is because *k*PTM uses a data level approach to send update messages while 2P2C uses a node level approach for maintaining consistency degree among cache nodes. As shown in Fig.12, the total traffic increases as update rate increases, this is because the increase in update rate produces a high number of forwarding messages in the network. In addition, the total traffic of *k*PTM is better than SPM and 2P2C methods. As shown in Fig.13, the query delay increases as update rate increases, this is because as update rate increases, the number of messages that will cross through the network increases which maximize the query delay for each crossing message. In addition, the query delay of SPM and *k*PTM is lower than 2P2C method, this is because the two proposed methods use data level approach while 2P2C uses node level approach. Also, the number of cache nodes with last update content in case of SPM and *k*PTM may be larger than 2P2C which can reply with the required data with small delay. As shown in Fig.14, the consistency degree increases as update rate increases, this is because sending more updating messages with high update rate can maintain the consistency of data items more accurately. In addition, the consistency degree of SPM and *k*PTM is much better than 2P2C method. This is because SPM and *k*PTM use data level policy while *2P2C* sends update messages based on normal path but with a node level. Also, the consistency degree of SPM is larger than *k*PTM. As shown in Fig.15, the server hit ratio, the local hit ratio and cache hit ratio are changeable as update rate increases. This is because the update rate does not effect on the hit ratio of data items.

5.3 Different number of data items

Fig.16, Fig.17, Fig.18, Fig.19, and Fig.20 show the communication overhead, total traffic, query delay, consistency degree, and hit ratio of SPM, *k*PTM, and 2P2C against different number of data items when the number of nodes was 30 nodes, the request rate was 2, the

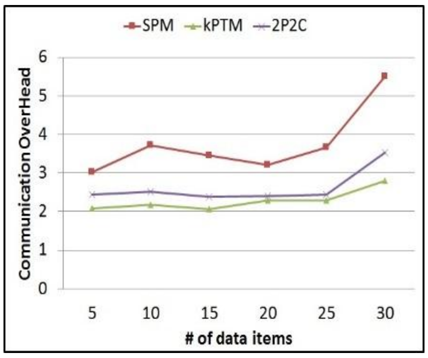

Fig. 16: Comm. overhead vs. # of data items

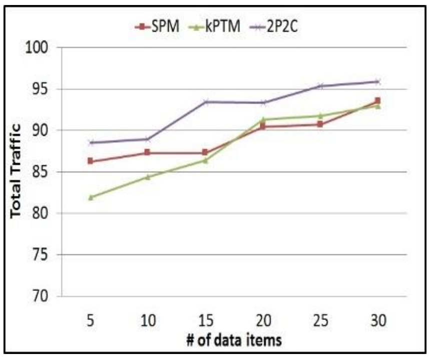

Fig. 17: Total traffic vs. # of data items

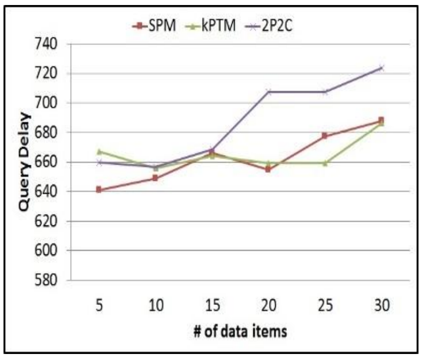

Fig. 18: Query delay vs. # of data items

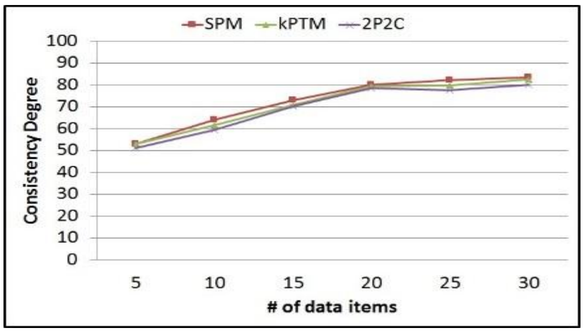

Fig. 19: Consistency degree vs. # of data items

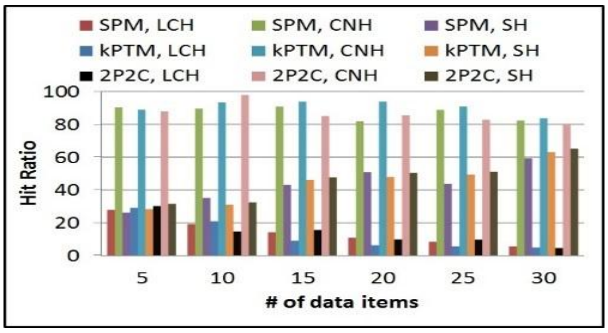

Fig. 20: Hit ratio vs. # of data items

update rate was 2, the cache size was 100MB, and the mobile node speed was 1m/s. of the proposed method SPM. As shown in Fig.16, the communication overhead increases as number of data items increases, this is because the growth in number of data items will increase the number of cache data items. Thus, a higher number of communication messages are needed to maintain the consistency degree for all cache data. In addition, the communication overhead of SPM is much larger than kPTM and 2P2C methods. This is because SPM uses a normal path from owner of a data item to a cache node while *k*PTM uses a tree structure to send update messages for all cache data and 2P2C sends update messages based on normal path but with a node level. Also, the communication overhead *k*PTM is lower than 2P2C method, this because *k*PTM sends update messages based on data level, while 2P2C uses a node level for maintaining consistency degree for all cache data. As shown in Fig.17, the total traffic increases as number of data items increases, this is because as number of data items increases, the number of forwarding and replying messages increases. In addition, the total traffic of the two proposed methods is much lower than 2P2C method and *k*PTM are better than 2P2C method. As shown in Fig.18, the query delay increases as number of data items increases, this is because the increase in number of data items will increase the number of updating, forwarding and replaying messages through the network which will affect the time delay for each query. In addition, the query delay of SPM and *k*PTM is lower than the query delay of 2P2C. As shown in Fig.19, the consistency degree increases as number of data items increases, this is because with a specific update rate, the increase in number of data items improves the average consistency degree among all data items in the network. In addition, as shown in Fig.19, the consistency degree of SPM and *k*PTM is larger than 2P2C method. This is because SPM uses data level approach while 2P2C sends update messages based on normal path, but with a node level approach. As shown in Fig.20, the local hit ratio decreases as number of data items increases, the cache hit ratio is changeable as number of data items increases, and the server hit ratio increases as number of data items increases. This is because when the number of data items increases the cache of each node does not keep each cache data item for long time. So, the number of replying messages by the owner of data item increases and it will receive a large number of requests. In addition, the local hit ratio and cache hit ratio of the proposed methods SPM and *k*PTM are better than 2P2C method.

5.4 Different speeds of a mobile node

Fig.21, Fig.22, Fig.23, Fig.24, and Fig.25 show the communication overhead, total traffic, query delay, consistency degree, and hit ratio of SPM, *k*PTM, and 2P2C against different values of mobile node speed when the number of data items was 10, the number of nodes was 30 nodes, the request rate was 2, the update rate was 2, and the cache size was 100MB. As shown in Fig.21, the communication overhead increases as mobile node speed increases from 1m/s to 4m/s while it decreases as mobile node speed increases from 5m/s to 6m/s. This is because with higher mobile node speed some of communication messages for updating cannot be received by cache nodes, so the number of communication messages will be decreased through the network. In addition, as shown in Fig.21, the communication overhead of SPM is much larger than *k*PTM and 2P2C methods. This is because SPM uses a normal path from owner of a data item to a cache node while *k*PTM uses a tree structure to send update messages for all cache nodes and 2P2C sends update messages based on normal path but with a node level. As shown in Fig.22, the total traffic increases as mobile node speed increases from 1m/s to 4m/s while it decreases as mobile node speed increases from 5m/s to 6m/. This is because with higher mobile node speed some of nodes cannot communicate to each other and cannot forward or reply to the requested nodes in the network. In addition, for lower values of mobile node speed, the two proposed methods achieve lower traffic than 2P2C. While for higher speed values, 2P2C achieves lower values of traffic than SPM and *k*PTM. As shown in Fig.23, the query delay increases as mobile node speed increases from 1m/s to 4m/s while it decreases as mobile node speed increases from 5m/s to 6m/s. This is because with higher values of mobile node speed, the number of requested queries will be small due to fast change in network topology which effect on the connection among nodes in the network. In addition, the query delay of SPM and *k*PTM is less than the query delay of 2P2C.

As shown in Fig.24, the consistency degree decreases as mobile node speed increases, this is because some of cache nodes cannot receive the update data message due to the dynamic change of network topology and some of network disconnections may exist. In addition, the consistency degree of SPM is larger than *k*PTM and 2P2C methods.

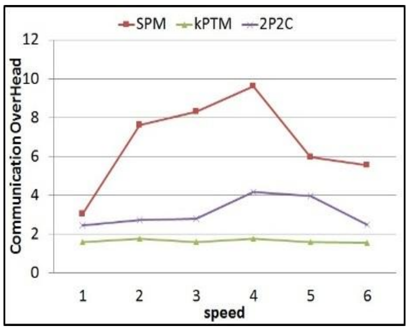

Fig. 21: Comm. overhead vs. speed

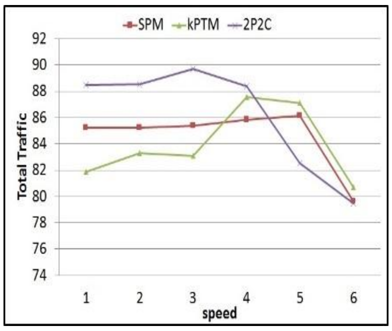

Fig. 22: Total traffic vs. speed

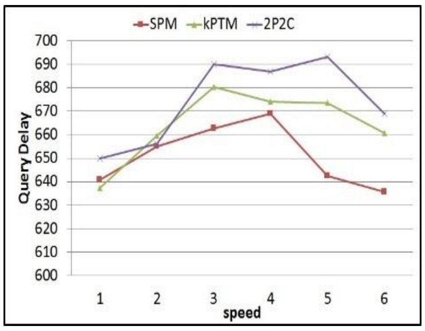

Fig. 23: Query delay vs. speed

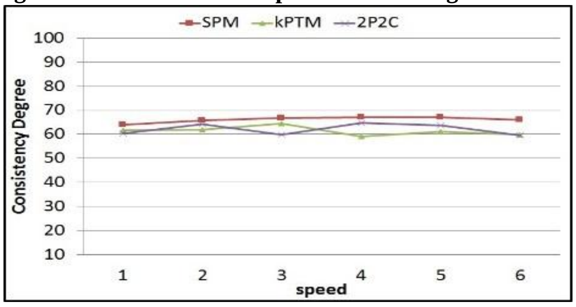

Fig. 24: Consistency degree vs. speed

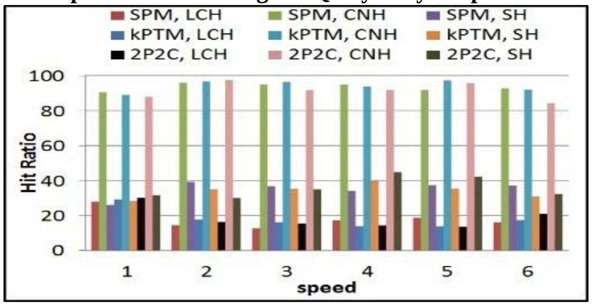

Fig. 25: Hit ratio vs. speed

This is because SPM uses a normal path from owner of a data item to a cache node which increases the success possibility to reach the destination cache nodes, while *k*PTM uses a tree structure which cannot be stable with some high-speed values, and 2P2C sends update messages based on normal path but with a node level (push or pull node). As shown in Fig.25, the server ratio increases as mobile node speed increases, while the local hit ratio and cache hit ratio decrease as request rate increases. this is because the increasing in mobile node speed will decrease the number of cache nodes for each data item and decrease the number of data items in a local cache of a node. In addition, the local hit ratio and cache hit ratio of the proposed methods SPM and *k*PTM are better than 2P2C method.

5.5 Different cache sizes

Fig.26, Fig.27, Fig.28, Fig.29, and Fig.30 show the communication overhead, total traffic, query delay, consistency degree, and hit ratio of SPM, *k*PTM, and 2P2C against different values of cache size when the number of data items was 10, the number of nodes was 30 nodes, the request rate was 2, the update rate was 2, and the mobile node speed was 1m/s. As shown in Fig.26, the communication overhead increases as cache size increases. This is because, for higher values of cache size in each node, the mobile node can store many data items, and they need to communicate with the data owner or the server to update their data items which will increase the number of messages to maintain the data consistency. In addition, as shown in Fig.26, the communication overhead of *k*PTM is much lower than SPM and 2P2C methods. This is because *k*PTM uses a tree structure to send update messages for all cache nodes, while SPM and 2P2C send update messages based on normal path with data level and node level policies, respectively. As shown in Fig.27, the total traffic increases as cache size increases from 50MB to 150MB, while it decreases as cache size increases from 200MB to 250MB. This is because for lower cache size, the mobile node can store few data items, so they will send many requests through the networks which maximize the number of forwarding and replying messages. While for higher cache size, the mobile node can several data items, so they do not need to send many requests through the networks which minimize the number of forwarding and replying messages. In addition, the total traffic for the two proposed methods is lower than 2P2C method. As shown in Fig.28, the query delay decreases as cache size increases from 50MB to 150MB while it decreases as cache size increases from 200MB to 250MB. This is because, for higher cache size, the mobile node can store several data items and they can find the required data item in its cache or in a nearby other cache node with small delay time. In addition, as shown in Fig.28, the query delay of SPM and *k*PTM methods is less than the query delay of 2P2C method. As shown in Fig.29, the consistency degree decreases as cache size increases, this is because the existence of a large number of different data items in each cache node which will effect on the whole consistency degree of all data items. In addition, as shown in Fig.29, the consistency degree of SPM and *k*PTM is larger than 2P2C method. As shown in Fig.30, the server ratio and the cache hit ratio decrease as cache size increases while the local hit ratio increases as request rate increases. This is because existence of a large cache size will maximize the local hit ratio for a node by fetching its local cache which stores several data items. In addition, the local hit ratio and cache hit ratio of the proposed methods SPM and *k*PTM are better than 2P2C method.

5.6 Different number of mobile nodes

Fig.31, Fig.32, Fig.33, Fig.34, and Fig.35 show the communication overhead, total traffic, query delay, consistency degree, and hit ratio of SPM, *k*PTM, and 2P2C against different number of mobile nodes when the number of data items was 10, the request rate was 2, the update rate was 2, the cache size was 100MB, and the mobile node speed was 1m/s. As shown in Fig.31, the communication overhead increases as number of mobile nodes increases, this is because the increasing in number of mobile nodes will increase the number of data requests for data items and will increase the number of cache nodes through the network. Thus, a higher number of communication messages will be transferred to maintain the consistency degree among all cache nodes.

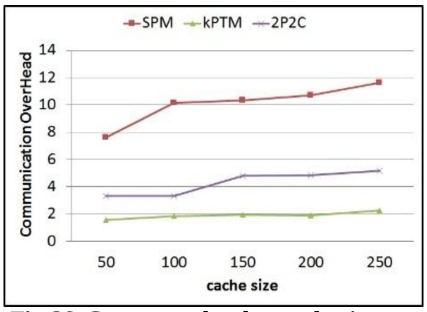

Fig. 26: Comm. overhead vs. cache size

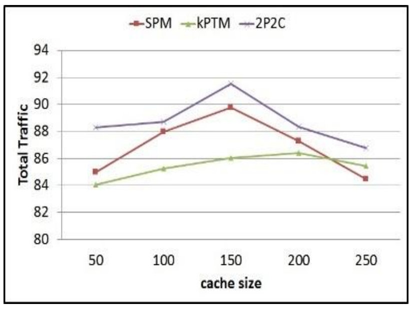

Fig. 27: Total traffic vs. cache size

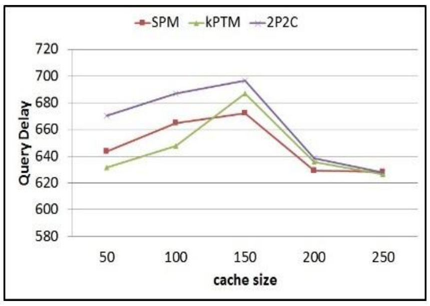

Fig. 28: Query delay vs. cache size

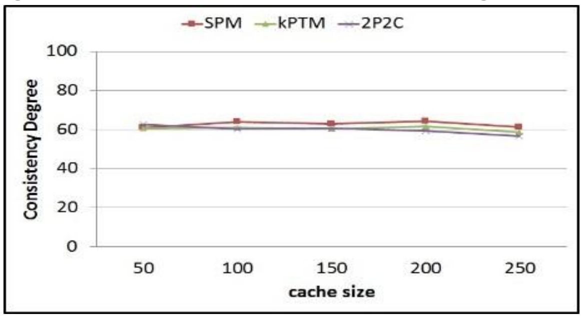

Fig. 29: Consistency degree vs. cache size

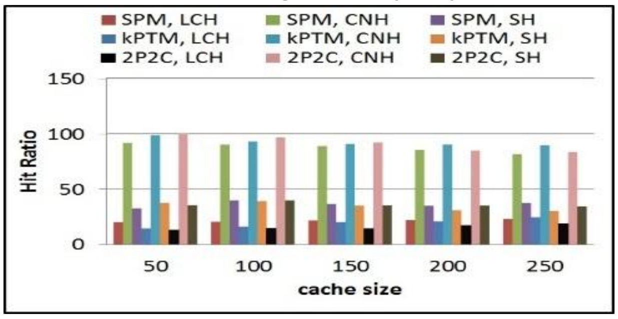

Fig. 30: Hit ratio vs. cache size

In addition, as shown in Fig.31, the communication overhead of SPM is much larger than *k*PTM and 2P2C methods. This is because SPM uses a normal path from owner of a data item to a cache node while *k*PTM uses a tree structure to send update messages for all cache nodes, and 2P2C sends update messages based on normal path but with a node level policy. Also, as shown in Fig.31, for higher values of number of mobile nodes, *k*PTM is better than 2P2C method, this because *k*PTM sends update messages based on data level policy, while 2P2C uses a node level for maintaining consistency degree among cache nodes. As shown in Fig.32, the total traffic increases as number of mobile nodes increases, this is because the increasing in number of mobile nodes will increase the number of cache nodes for each data item and increase the number of forwarding and replying messages in the network. In addition, the total traffic for the *k*PTM is lower than SPM and 2P2C methods. As shown in Fig.33, the query delay increases as number of mobile nodes increases from 15 to 30 nodes while it decreases when number of mobile nodes increases from 45 to 75 nodes. This is because if there is a low number of mobile nodes, a small number of messages will be generated through the network which will not affect the time delay for each query while if there is a high number of mobile nodes, many messages through the network will be generated which will effect on the time delay for each query. In addition, as shown in Fig.33, the query delay of SPM and *k*PTM is less than the query delay of 2P2C. As shown in Fig.34, the consistency degree decreases as the number of mobile nodes increases, this is because the increasing in number of mobile nodes will increase the number of cache nodes in the network for each data item. Thus, a higher number of updating messages are needed to maintain the consistency degree among all cache nodes. In addition, as shown in Fig.34, the consistency degree of SPM and *k*PTM is larger than 2P2C method. This is because SPM and *k*PTM use a data level for updating while 2P2C sends update messages based on normal path but with a node level. As shown in Fig.35, the server ratio decreases as number of mobile nodes increases while the cache hit ratio increases as request rate increases. But the local hit ratio is almost the same. This is because with more mobile nodes in the network, there is a possibility to increase the number of cache nodes which will maximize the cache hit ratio and minimize the server hit ratio. In addition, the cache hit ratio of the proposed SPM and *k*PTM are better than 2P2C method in almost cases.

5.7 Different values of parameter k

Fig.36, Fig.37, Fig.38, Fig.39, and Fig.40 show the communication overhead, total traffic, query delay, consistency degree, and hit ratio of SPM, *k*PTM, and 2P2C against different split values of *k* for *k*PTM which are 1, 2, 3, 4, 5, 6 when the number of data items was 10, the number of mobile nodes was 30, the request rate was 2, the update rate was 2, the cache size was 100MB, and the mobile node speed was 1m/s. As shown in Fig.36 and Fig.37, the communication overhead and the total traffic increase as split value increases, this is because when split value increases *k*PTM tends to be as SPM with normal paths. As shown in Fig.38, the query delay decreases as split value increases, this is because the increasing in split value will increase the number of paths in the tree which will effect on the time delay of each query. As shown in Fig.39, the consistency degree increases as split value increases, this is because *k*PTM tends to be as SPM with normal paths and update data will be sent directly to each cache node. As shown in Fig.40, the server ratio decreases as split value increases while the cache hit ratio increases as split value increases. But the local hit ratio is less unchangeable by increasing the split value. This is because *k*PTM will work as SPM method to send update data for all cache nodes.

6. CONCLUSION

In this paper, a new adaptive hybrid data-based cache consistency scheme is proposed called A2P2C scheme. The proposed scheme classifies the data items into push data items and pull data items based on the owner decision. A2PD2C proposed two new updating methods SPM method and *k*PTM method to improve the cache consistency. The conducted simulation results showed that the proposed A2PD2C can maintain the data consistency, decrease unnecessary communication overhead, reduce access latency, reduce the server hit ratio, maximize the cache and local hit ratios, and was much better than existing methods. In the future work, the optimal tree structure and the optimal value of *k* for *k*PTM will be studied to improve the consistency degree.

7. ACKNOWLEDGMENTS

I feel greatly privileged to express thanks to IJCA for giving me this opportunity to publish this work.

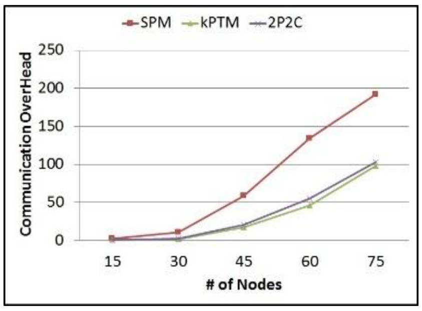
Fig. 31: Comm. overhead vs. # of nodes

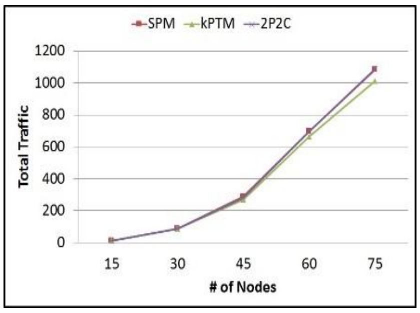
Fig. 32: Total traffic vs. # of nodes

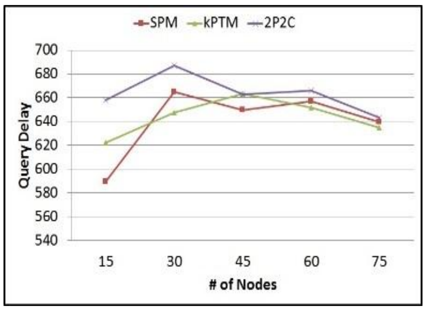
Fig. 33: Query delay vs. # of nodes

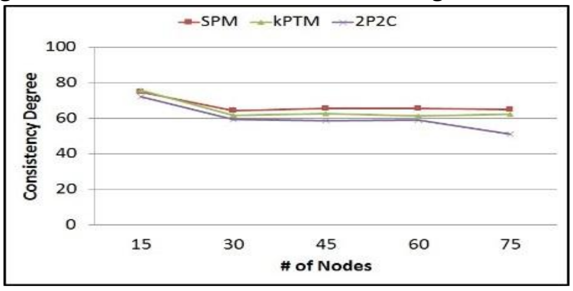
Fig. 34: Consistency degree vs. # of nodes

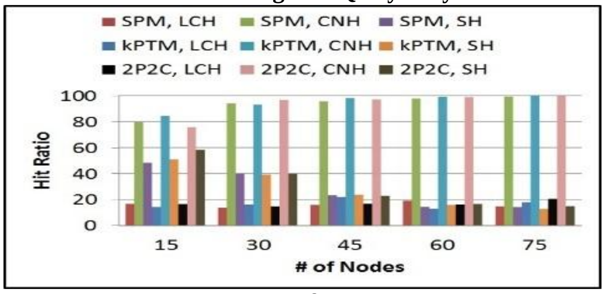
Fig. 35: Hit ratio vs. # of nodes

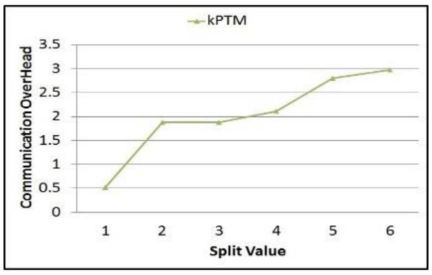
Fig. 36: Comm. overhead vs. split value

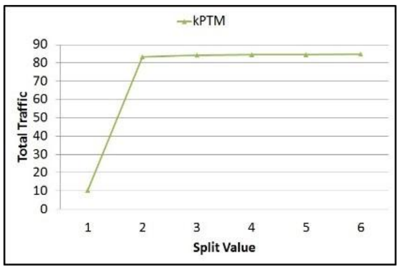
Fig. 37: Total traffic vs. split value

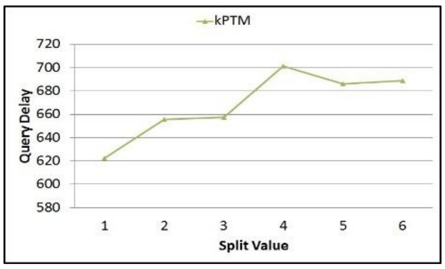
Fig. 37: Query delay vs. split value

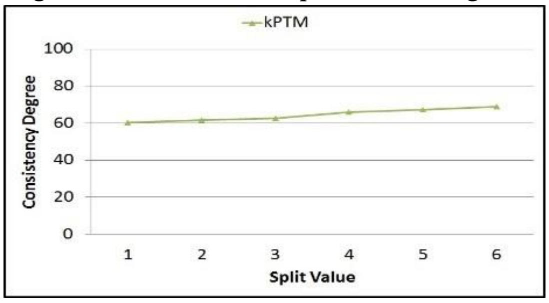
Fig. 38: Consistency degree vs. split value

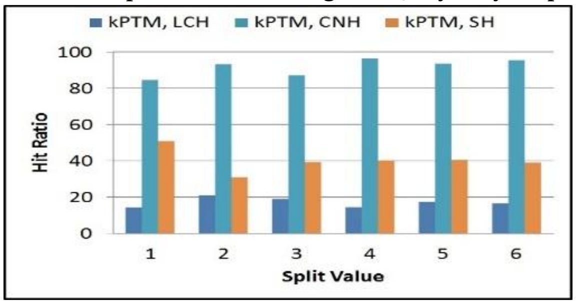
Fig. 40: Hit ratio vs. split value

8. REFERENCES

[1] Yin L., Cao G. (2006). Supporting cooperative caching in ad hoc networks. IEEE Transactions on Mobile Computing, 5, 77-89.

[2] Lim S., Lee W., Cao G., Das C. (2006). A novel caching scheme for improving internet based mobile ad hoc networks performance. Ad Hoc Networks, 4, 225-239.

[3] Artail H., Safa H., Mershad K., Abou-Atme Z., Sulieman N. COACS. (2008). A cooperative and adaptive caching system for MANETs. IEEE Transactions on Mobile Computing, 7, 961-977.

[4] Krishnamurthy B., Wills CE. (1998). Piggyback server invalidation for proxy cache coherency. In Seventh International World-Wide Web Conference (pp. 185-193), Brisbane, Australia.

[5] Krishnamurthy B., Wills C. (1997). Study of piggyback cache validation for proxy caches in the World Wide Web. In Usenix Symposium on Internet Technologies and Systems (pp. 1-12), California, USA.

[6] Yin L., Cao G., Cai Y.A. (2003). Generalized target driven cache replacement policy for mobile environments. In International Symposium on Applications and the Internet (pp. 14-21), Orlando, FL, USA.

[7] Fawaz K., Artail H. (2013). DCIM: Distributed cache invalidation method for maintaining cache consistency in wireless mobile networks. IEEE Transactions on Mobile Computing, 12, 680-693.

[8] Mershad K., Artail H. (2010). SSUM: Smart server update mechanism for maintaining cache consistency in mobile environments. IEEE Transactions on Mobile Computing, 9, 778-795.

[9] Cao J., Zhang Y., Cao G., Li X. (2007). Data consistency for cooperative caching in mobile environments. Computer, 40, 60-66.

[10] Cao P., Liu C. (1998). Maintaining strong cache consistency in the world-wide web. IEEE Transactions on Computers, 47, 445-457.

[11] Jing J., Elmagarmid A., Helal A., Alonso R. (1997). Bit-sequences: an adaptive cache invalidation method in mobile client/server environments. Mobile Networks and Applications, 2, 115-127.

[12] Tang X., Xu J., Lee WC. (2008). Analysis of TTL-based consistency in unstructured peer-to-peer networks. IEEE Transactions Parallel and Distributed Systems, 19, 1683-1694.

[13] Jung J., Berger AW., Balakrishnan H. (2003). Modeling TTL-based internet caches. In INFOCOM (pp. 417-426), San Francisco, CA, USA.

[14] Cao G. (2003). A scalable low-latency cache invalidation strategy for mobile environments. IEEE Transactions on Knowledge and Data Engineering, 15, 1251-1265.

[15] Selvin , L. S., palanichamy, Y. (2016): Push-pull cache consistency mechanism for cooperative caching in mobile ad hoc environments. Turkish Journal of Electrical Engineering and Computer Sciences, 24(5), 3459-3470.

[16] Joy, P. T., Jacob, K. P. (2012): A Comparative Study of Cache Replacement Policies in Wireless Mobile Networks. Advances in Computing and Information Technology, Springer Berlin Heidelberg, 176, 609-619.

[17] Dar S., Franklin M. J., Jonsson B. T., Srivastava D., Tan M. (1996): Semantic data caching and replacement. In Proceedings of the 22th International Conference on Very Large Data Bases (VLDB) (pp. 330-341).

[18] Ren Q., Dunham M. H. (2000): Using semantic caching to manage location dependent data in mobile computing. In Proceedings of the 6th ACM annual international conference on Mobile computing and networking (MOBICOM) (pp. 210-221), Boston, MA, USA.

[19] Lai K. Y., Tari Z., Bertok P. (2004): Mobility-aware cache replacement for users of location-dependent services. In the 29th Annual IEEE International Conference on Local Computer Networks (pp. 50-58).

[20] ElDahshan K.A., Ahmed A.A.G., Sobhi A., (2015): A Distance-based Predicted Region Policy for Cache Replacement in Mobile Environments. International Journal of Computer Applications, 126, 1-10.

[21] Ahmed A.A.G., ElDahshan K.A., Sobhi A. (2016): A Predictable Markov Based Cache Replacement Scheme in Mobile Environments. International Journal of Computer Science and Information Security, 14(4), 15-26.

[22] Varga, András. (2001). "The OMNeT++ discrete event simulation system." Proceedings of the European simulation multi-conference (ESM'2001).