J. Parallel Distrib. Comput. 71 (2011) 1034–1046

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

# Adaptive per-user per-object cache consistency management for mobile data access in wireless mesh networks

Yinan Li *, Ing-Ray Chen

*Department of Computer Science, Virginia Tech, Northern Virginia Center, 7054 Haycock Road, Falls Church, VA 22043, United States*

ARTICLE INFO

*Article history:*
Received 9 August 2010
Received in revised form
27 December 2010
Accepted 11 March 2011
Available online 21 March 2011

*Keywords:*
Cache consistency management
Mobile data access
Data proxy
Wireless mesh networks
Performance analysis

ABSTRACT

We propose and analyze an adaptive per-user per-object cache consistency management (APPCCM) scheme for mobile data access in wireless mesh networks. APPCCM supports strong data consistency semantics through integrated cache consistency and mobility management. The objective of APPCCM is to minimize the overall network cost incurred due to data query/update processing, cache consistency management, and mobility management. In APPCCM, data objects can be adaptively cached at the mesh clients directly or at mesh routers dynamically selected by APPCCM. APPCCM is adaptive, per-user and per-object as the decision regarding where to cache a data object accessed by a mesh client is made dynamically, depending on the mesh client's mobility and data query/update characteristics, and the network's conditions. We develop analytical models for evaluating the performance of APPCCM and devise a computational procedure for dynamically calculating the overall network cost incurred. We demonstrate via both model-based analysis and simulation validation that APPCCM outperforms non-adaptive cache consistency management schemes that always cache data objects at the mesh client, or at the mesh client's current serving mesh router for mobile data access in wireless mesh networks.



## 1. Introduction

Wireless mesh networks (WMNs) are emerging in recent years as a promising standard for next-generation broadband wireless networks and are regarded as a cost-effective solution for providing last-mile broadband wireless Internet connectivity [19]. A WMN consists of two types of components: wireless mesh routers (MRs) and mesh clients (MCs) [1]. Each MR serves as both a router that forwards packets and a wireless access point for MCs. The group of MRs within a WMN forms a wireless mesh backbone that provides last-mile broadband Internet connectivity to MCs. A WMN is interconnected to the Internet through the gateway functionality of MRs. MCs are devices that have wireless access capability, e.g., laptops, smartphones, PDAs, etc.

In this paper, we investigate the problem of data caching and cache consistency management for mobile data access in WMNs. Mobile data access, which is fundamental to client–server computing in mobile environments [11], is challenging due to the intrinsic characteristics of mobile computing: mobility and resource constraints of clients, and bandwidth constraints of wireless communications [9,20]. An additional challenge as a result of these characteristics is that mobile data access must cope with voluntary or involuntary disconnection of mobile clients. That is, mobile data access must support disconnected operations [13]. Caching is a key technique for improving the performance of mobile data access because it alleviates constraints such as highly variable mobile connectivity and wireless bandwidth limitations, and it significantly reduces the latency for answering a query. Caching is also the basis for disconnected operations in mobile data access.

Data caching is particularly beneficial for mobile Internet data access in WMNs for a major reason. Specifically, because WMNs are a cost-effective solution for last-mile broadband Internet access, Internet traffic contributed considerably by mobile Internet data access is expected to dominate network traffic in WMNs. Because Internet traffic always passes through the gateway, the gateway is potentially the bottleneck under conditions of heavy Internet traffic and network congestion is highly probably to happen around the gateway. Caching is an efficient solution for mitigating the problem because it can significantly reduce the number of uplink and downlink messages passing through the gateway in client–server mobile Internet data access, thereby mitigating the performance bottleneck at the gateways in WMNs [6].

A central issue closely related to caching in mobile data access is cache consistency management because clients may not be able to keep cached data synchronized due to mobility and disconnection. We propose and analyze an adaptive per-user per-object cache consistency management (APPCCM) scheme for mobile data access in WMNs. APPCCM supports strong data consistency semantics through integrated cache consistency and mobility management.

* Corresponding author.
*E-mail addresses:* yinan926@vt.edu (Y. Li), irchen@vt.edu (I.-R. Chen).

0743-7315/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2011.03.001

The objective of APPCCM is to minimize the overall network cost incurred collectively by data query/update processing, cache consistency management, and mobility management. APPCCM provides two data access and caching modes: a data object can be cached either directly at the MC, or at a *data proxy* running on an MR dynamically selected by APPCCM. We use the terms *client-cache mode* (CCM) and *data-proxy mode* (DPM) to refer to these two modes throughout the paper. CCM is based on existing asynchronous stateful-based cache invalidation schemes [2,21] augmented with the capability of integrated cache consistency and mobility management such that MCs can perform data access, data caching and cache consistency management while roaming in a WMN. DPM is distinct from traditional approaches by exploiting the capability of increasingly powerful MRs to perform data caching in addition to routing. The rationale of using data proxies is that it is beneficial to cache a data object at the data proxy rather than at the MC under certain operational and environmental conditions which we aim to identify in the paper.

APPCCM is adaptive, per-user and per-object because for each individual MC, the decision of where to cache data objects is made dynamically and independently for each object based on the MC's mobility and data query/update characteristics, and the WMN conditions. We develop a computational procedure for dynamically calculating the overall communication cost in APPCCM, given parameters characterizing the MC's mobility and data query/update characteristics, and the WMN conditions. For DPM, APPCCM determines *when* a cached data object should be migrated between two data proxies due to MC mobility. We use a threshold on the number of location changes of the MC denoted by $K_{\text{optimal}}$ to model the optimal time point at which the data object should be migrated such that the overall communication cost is minimized. When the threshold is reached, the data object will be migrated to the data proxy on the MC's current serving MR.

We develop analytical models for evaluating the performance of APPCCM and devise a computational procedure for dynamically calculating the overall network cost incurred. We demonstrate via both model-based analysis and simulation validation that APPCCM outperforms non-adaptive cache consistency management schemes that always cache data objects at the mesh client, or at the mesh client's current serving mesh router for mobile data access in wireless mesh networks.

The rest of the paper is organized as follows. Section 2 surveys existing work and contrasts our scheme with existing schemes. Section 3 presents the system model and assumptions made in the paper. The proposed APPCCM scheme is presented in Section 4. In Section 5, we develop mathematical models for evaluating APPCCM. Performance analysis and comparison are carried out in Section 6. The paper concludes with Section 7.

## 2. Related work

The issue of cache consistency management in wireless data access has been intensively studied. Most existing work focuses on cache invalidation strategies. The basic idea of cache invalidation for cache consistency management in wireless data access proposed in [2] is as follows: the server periodically broadcasts *invalidation reports* (IRs), which carry information about data objects that have been updated by the server in the most recent time interval. Clients invalidate obsolete cached data objects according to the content of invalidation reports. Based on this basic approach, many different IR-based cache invalidation schemes have been proposed in the literature [10,8,3,12,4,16,22,23,18].

These schemes fall into two categories: *stateless-based* schemes and *stateful-based* schemes [21]. Stateless-based approaches are very popular and most existing schemes fall into the stateless-based category because the server does not need to keep state information and the server-side overhead is minimum [21]. In stateless-based approaches, the server either synchronously or asynchronously broadcasts IRs for updated data objects. In synchronous stateless-based approaches, IRs are broadcast periodically by the server, whereas in asynchronous stateless-based approaches, they are broadcast whenever data objects are updated. Asynchronous stateless-based approaches allow connected clients to keep their cache contents synchronized instantaneously. A disconnected client, however, may need to discard the entire cache contents because it has no idea about which data objects have been updated during its disconnection. Synchronous stateless-based approaches strike a balance in waiting time for both connected and disconnected clients because the waiting time for the next IR is bounded by the broadcast interval. However, synchronous stateless-based approaches have the drawback that they consume wireless bandwidth significantly for broadcasting the IRs.

In stateful-based approaches, the server is stateful as it keeps information about where data objects are cached. The *callback* (CB) algorithm [24,16] uses the stateful-based approach to maintain strong cache consistency. In the CB approach, whenever the server updates a data object, it asynchronously sends an IR to each client that keeps a cached copy of the updated data object. Upon receiving the IR and removing the obsolete data object, the client sends the server an acknowledgement confirming that the invalidation is successful. If the server subsequently updates the data object before the client queries it for the first time after the invalidation, no IR needs to be sent to the client. To answer a query, the client checks its cached copy to see if it has been invalidated. If there is a valid copy, the query is answered immediately. Otherwise, the client sends the query to the server to retrieve a fresh copy of the data object and stores it into the cache before answering the query. Stateful-based approaches have the same drawback as in asynchronous stateless-based approaches, i.e., a disconnected client may need to discard the entire cache contents upon reconnection because it has no idea about which data objects have been updated during its disconnection.

*Poll-each-read* (PER) [24,16] maintains strong cache consistency by always checking if the queried data object is still valid in the cache for every query. The PER approach is not based on cache invalidation. Specifically, in the PER approach, the server does not send IRs when data objects are updated. Instead, to answer a query, a client always sends a message to the server to check if the cached copy of the queried data object is still valid. If the queried data object is still valid in the cache, the server sends the client an affirmative message, and the client retrieves the cached copy of the data object to answer the query. If the data object has been updated before the query, the server sends the updated data object to the client, and the client stores the data object into the cache before answering the query. Therefore, a cache hit in the PER approach is not as beneficial as in approaches based on cache invalidation. The PER approach avoids the signaling overhead for transmitting IRs with the cost of increasing the average latency for answering a query. This algorithm incurs unnecessary signaling overhead and delay when a data object is queried more frequently than being updated by the server.

Although many cache consistency management schemes exist, these existing schemes generally cannot be used directly for cache consistency management in WMNs without considerable modification and performance penalty. Existing stateless-based approaches are not appropriate for WMNs because they rely on the gateway (as a server surrogate) to broadcast invalidation reports, so the gateway can easily become the bottleneck when data objects are frequently updated by the data server. Existing stateful-based approaches incur too much overhead to track the locations of MCs to deliver invalidation reports or data objects to roaming MCs because of a lack of cache consistency management integrated

with user mobility support in WMNs. Another limitation of existing schemes is that most of them consider read-only mobile data access, making them not appropriate for applications supporting client-side updates that are propagated to the server.

In this paper we propose APPCCM to address the limitations of existing schemes. APPCCM is based on integrated cache consistency management and user mobility support as it minimizes the overall network cost incurred by data query/update processing, cache consistency management, and mobility management. APPCCM supports both read and write operations to data objects. Moreover, APPCCM is specifically designed for WMNs, taking into consideration the characteristics of WMNs. We leverage the fact that modern MRs have adequate processing power and expandable memory capacity (via USB-based flash or hard drives) [6], making it feasible to perform caching at data proxies running on the MRs. This work extends [14] with a refined design of APPCCM with model-based analysis, a more thorough performance comparative study with existing non-adaptive schemes, including simulation validation of analytical results and sensitivity analysis of design parameters, as well as a discussion of the applicability and practicality of APPCCM.

## 3. System model

In this paper, we consider a scenario in which MCs within a WMN access data objects on a data server that is located outside of the WMN but is accessible to the WMN through a wired connection between the server and the gateway. Because the server is not within the WMN, all data queries must go through the gateway, thus incurring a substantial amount of traffic load onto the gateway. To alleviate the performance bottleneck at the gateway and make data access more efficient, each MC maintains a cached copy for each queried data object either in its local cache or at a data proxy running on an MR dynamically selected by APPCCM, which may be the MC's current serving MR or one of its previous serving MRs. In certain circumstances, it is beneficial to cache a data object directly at the MC, whereas in other circumstances it is beneficial to cache a data object at a data proxy. The decision of where to cache a data object is adaptively made based on the MC's data query/update and mobility characteristics.

APPCCM is based on a stateful approach by which when a data object is updated, an IR is asynchronously sent from the server to the MCs and data proxies that keep a cached copy of the data object. Recall that the server is stateful as it keeps state information about which clients cache which data objects. Specific to this work, the server sends an IR to the gateway whenever it updates a data object, and the gateway forwards the IR to those MCs and data proxies that keep a cache copy of the data object. Therefore, the gateway rather than the server is stateful and keeps information about where data objects are cached. We assume that the gateway always keeps a copy of every data object accessed by MCs within the WMN. Therefore, the gateway is like a replica of the server.

The data query/update characteristics between an MC and the server are specified by a number of parameters. We assume that the inter-arrival time between two consecutive queries to the same data object from the same MC follows a Poisson distribution with mean $1/\lambda$, where $\lambda$ is the query rate. Updates to the data object by the server also follow a Poisson distribution with mean $1/\mu$, where $\mu$ is the rate of updates by the server. Additionally, updates to the data object from MCs follow Poisson distributions. Here we need to differentiate between two different cases: local updates from the MC under consideration and remote updates from other MCs within the WMN. We use $\delta$ and $\eta$ to denote the rate of local updates by the MC under consideration and the aggregate rate of remote updates by other MCs in the WMN, respectively. We define a parameter called *query to update ratio* (QUR) to model the data query/update characteristics between the MC and the server. The QUR is given by: $\text{QUR} = \frac{\lambda}{\mu+\delta+\eta}$.

An MC may voluntarily disconnect from the WMN and switch to idle mode periodically to reduce power consumption during its stay within the network. The MC may also involuntarily disconnect due to a weak wireless connection. We view MCs that are disconnected from the WMN as being in idle mode, regardless of the reasons for disconnection, as a disconnected MC cannot transmit nor receive any data, therefore incurring no network cost. We model the transition between active mode and idle mode using three parameters: $\omega$, $\omega_w$, and $\omega_s$. The physical meaning of $\omega$ is the rate of reconnection of an MC with the WMN given that the MC is in idle mode. The reciprocal of $\omega_w$ indicates the average duration of disconnection of the MC before a transition from idle mode to active mode. Similarly, The reciprocal of $\omega_s$ denotes the average duration in which the MC keeps connected before a transition from active mode to idle mode.

We assume that future mobile devices, e.g., PDAs, smartphones, tablet computers, etc., are powerful enough to execute the computational procedure developed in this paper. It is worth emphasizing that the computational procedure developed in the paper is lightweight and just needs to be executed periodically by a mobile device to determine its operational settings to minimize the network traffic. The end result is minimized communication costs not only to the WMN as a whole but also to individual mobile devices in the network. As energy consumption due to computation is small compared with that due to communication, we believe that the energy saving from minimizing the communication costs outweighs the energy consumption from executing the lightweight computational procedure.

## 4. The proposed APPCCM scheme

### 4.1. DPM vs. CCM

There are two caching modes in APPCCM, namely CCM and DPM. In CCM, a data object accessed by an MC is cached directly by the MC, whereas in DPM, the data object is cached by a data proxy running on an MR. A data proxy is essentially a data cache maintained by an MR. Modern MRs have sufficient computing power and storage capacity to perform both routing and data caching [6]. The rationale of using data proxies to cache data objects is that it incurs less network cost than always caching data objects directly at the MCs, under certain circumstances. More specifically, when a data object is updated more frequently than being accessed by an MC such that the invalidation cost is dominating, it may be beneficial to cache the data object at a data proxy rather than locally at the MC to reduce the invalidation cost and hence the total communication cost. On the other hand, if a data object is accessed more frequently by the MC than being updated such that the access cost is dominating, it may be beneficial to let the MC cache the data object directly to avoid the additional cost of accessing a data proxy. Therefore, there exists a tradeoff between the access cost and invalidating cost. APPCCM exploits this tradeoff and adaptively decides on a per-user per-object basis where to cache a data object based on the data object's QUR and the MC's mobility characteristic. The decision is made independently for each data object accessed by each individual MC. Therefore APPCCM is an adaptive per-user per-object cache consistency management scheme.

### 4.2. Data access and caching

In APPCCM, each MC maintains a *caching status table* that keeps the caching status of each data object it has accessed. The caching

**Table 1**
The fields of a caching status table and explanations.

| Field | Explanation |
|---|---|
| Object ID | The data object identifier; each data object has a unique identifier |
| Caching location | The location where the data object is cached |
| Address of data proxy | The address of the data proxy where the data object is cached if applicable |
| Time stamp | The time stamp when the cached copy of the data object is most recently updated due to query |

status table has four fields as shown and explained in Table 1. When an MC receives a new data query from an application, it first checks its caching status table to see whether an entry for the queried data object exists or not in the table, and if an entry is found, it determines where the data object is currently cached and if the cached copy is still valid. Depending on the result of this table lookup, the query is answered accordingly in different ways. The pseudo code presented below describes the query processing algorithm.

**Algorithm 1** The query processing algorithm.

```
if an entry is not found then
    the MC sends the query to the server to retrieve a fresh copy
    of the data object;
    if CCM is to be used to cache the object then
        the MC puts the received data object into its local cache upon
        receiving it;
    else
        upon receiving the data object, the MC's current serving MR
        puts it into the data proxy before forwarding it to the MC;
    end if
    the MC updates its caching status table;
else
    if the data object is found cached by the MC then
        if the cached copy is still valid then
            the query is answered immediately locally;
        else
            the MC sends the query to the server, and upon receiving
            the data object, the MC updates the cached copy and the
            caching status table;
        end if
    else
        the MC sends the query to the data proxy specified in the
        caching status table;
        if the cached copy is still valid then
            the data proxy sends the data object to the MC;
        else
            the data proxy forwards the query to the server, and upon
            receiving the data object, updates the cached copy and
            forwards the data object to the MC;
        end if
        upon receiving the data object, the MC updates the caching
        status table;
    end if
end if
```

In this paper, we consider that in addition to the server, an MC can also update any data object for which it keeps a cached copy. Therefore, MCs have both read and write permissions. Whenever an MC updates a data object, it sends the updated object and an IR to the gateway, which forwards the updated data object to the server and, upon receiving an acknowledgement from the server that the update is accepted, forwards the IR to those MCs and data proxies that keep a cached copy of the data object to be invalidated.

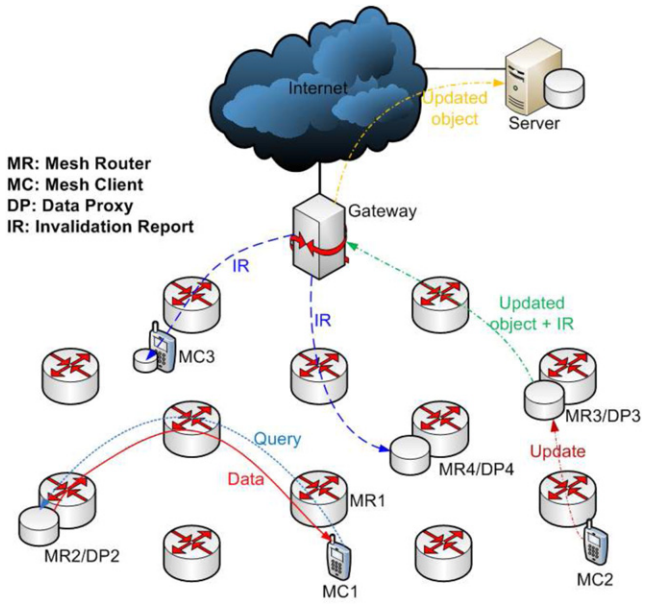



**Fig. 1.** Examples of query and update processing in APPCCM.

Fig. 1 illustrates examples of query and update processing in APPCCM. As the figure shows, MC1 that is connected to MR1 employs MR2 as a data proxy to cache some data objects it has accessed. Upon receiving a query for a data object cached in the data proxy running on MR2, MC1 sends the query to MR2, which sends the queried data object back to MC1. In another example, MC2 updates a data object cached in the data proxy running on MR3. After the update is completed, MR3 sends the updated object and an IR to the gateway, which then forwards the updated data object to the server. Upon receiving an acknowledgement from the server that the update is accepted, the gateway forwards the IR to those MCs and data proxies that keep a cached copy of the data object being invalidated, namely, MC3 and MR4 in this example.

### 4.3. MC mobility and data migration

We explicitly consider the effect of mobility of MCs on data caching and cache consistency management in APPCCM. MC mobility affects the two caching modes of APPCCM differently. Specifically, in DPM, a threshold denoted by $K$ is specified for each data object accessed by an MC, such that when the distance between the MC's current serving MR and the data proxy where the data object is cached reaches $K$ due to MC mobility, the data object is migrated to the data proxy on the MC's current serving MR. The optimal threshold $K_{\text{optimal}}$ that results in minimized total communication cost is determined dynamically on a per-user per-object basis.

DPM can be easily implemented, as illustrated by Fig. 2. Specifically, each MC keeps a counter for each data object that records the number of location changes of the MC since the data object's most recent migration. Each time when the MC moves and changes its serving MR, the counter is incremented by 1. When the counter reaches the threshold $K$ after a movement, the data object will be migrated from its current data proxy to the one running on the new serving MR of the MC. After the data migration, the counter is reset to zero.

In DPM, there are chances that the same MR is chosen by more than one MC as the data proxy for caching the same data object. In this case, only a single cached copy of the data object needs to be kept by the MR serving all such MCs. For each such shared data object, the data proxy maintains a count indicating the number

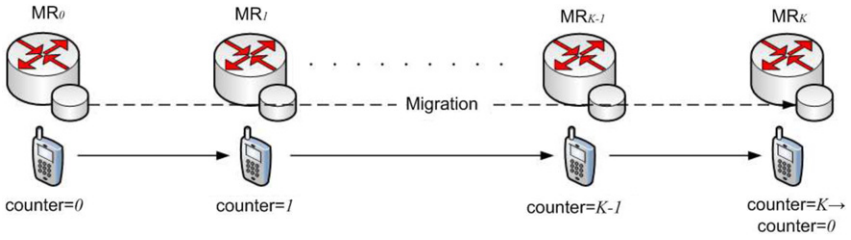



**Fig. 2.** Threshold-based data object migration in DPM.

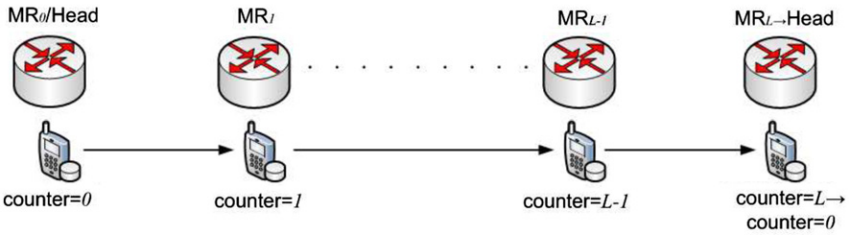



**Fig. 3.** Pointer forwarding based location management in CCM.

of MCs that share the data object. When the count reaches zero because of object migration, the data object can be deleted from the data proxy.

In CCM, a location management scheme is employed to track the locations of MCs in order for the gateway to deliver IRs to the destination MCs. We develop a per-user location management scheme for CCM based on pointer forwarding [5,15]. In this scheme, the gateway maintains a location database where the address of the forwarding chain head of each MC is kept. A threshold denoted by $L$ is specified for each MC to regulate the allowable forwarding chain length. The optimal threshold $L_{\text{optimal}}$ under which the overall communication cost for location management and service delivery is minimized is dynamically determined for each MC based on the MC's mobility and service characteristics. When an MC moves and changes its serving MR, a forwarding pointer is setup between the two involved MRs, and the forwarding chain length is increased by 1. When the forwarding chain length of the MC reaches the threshold $L$, its current forwarding chain is reset and the new serving MR becomes its new forwarding chain head. A location binding update message is sent to the gateway in this case to update the location information of the MC, i.e., the address of its forwarding chain head.

Fig. 3 illustrates how the location management scheme in CCM is implemented. Like in DPM, each MC maintains a counter that records the number of movements since the most recent location update. Each time the MC moves to a new MR, the counter is incremented by 1, indicating that the length of the forwarding chain increases by 1. When the counter reaches the threshold $L$ after a movement, a location update is performed and the MC's current forwarding chain is reset. After the location update, the new serving MR becomes its new forwarding chain head, and the counter is reset to zero.

## 5. Analytical modeling

In this section, we develop analytical models based on stochastic Petri net (SPN) for analyzing the performance of APPCCM. Table 2 lists the parameters used in performance modeling and analysis.

### 5.1. SPN models for APPCCM

Fig. 4 presents the SPN model for DPM. The meanings of places and transitions in the SPN model are defined in Table 3. `Mark(P)` returns the number of tokens in place P. In the SPN model, `#(P)` associated with an arc means that the multiplicity of the arc is equal to the number of tokens in place P. The SPN model for DPM is constructed as follows:

- The event of MC movement is modeled by transition *Move*, the transition rate of which is $\sigma$. When an MC moves to and is

**Table 2**
Parameters used in performance modeling and analysis.

| Parameter | Meaning |
|---|---|
| $\sigma$ | Mobility rate of an MC |
| $\lambda$ | Rate of queries from an MC for a data object |
| $\mu$ | Rate of updates of the server to a data object |
| $\delta$ | Aggregate rate of updates of all MCs but the one under consideration to a data object |
| $\eta$ | Rate of updates of the MC under consideration to a data object |
| $\omega$ | Rate of reconnection when an MC switches from idle mode back to active mode |
| $\omega_w$ | $\frac{1}{\omega_w}$ indicates the average duration of disconnection of the MC before a transition from idle mode to active mode |
| $\omega_s$ | $\frac{1}{\omega_s}$ denotes the average duration within which the MC keeps connected before a transition from active mode to idle mode |
| QUR | Query to update ratio, defined as $\frac{\lambda}{\mu+\delta+\eta}$ |
| QMR | Query to mobility ratio, defined as $\frac{\lambda}{\sigma}$ |
| $P_{\text{active}}$ | Probability of an MC in active mode, defined as $\frac{\omega_w}{\omega_w+\omega_s}$ |
| $\alpha$ | Average distance (number of hops) between the gateway and an arbitrary MR |
| $\beta$ | Average distance (number of hops) between the current serving MR of an MC after it reconnects and its serving MR before disconnection |
| $\tau$ | One-hop communication cost between two MRs |
| $P_{\text{hit}}$ | Cache hit ratio |
| $P_{\text{miss}}$ | Cache miss ratio |

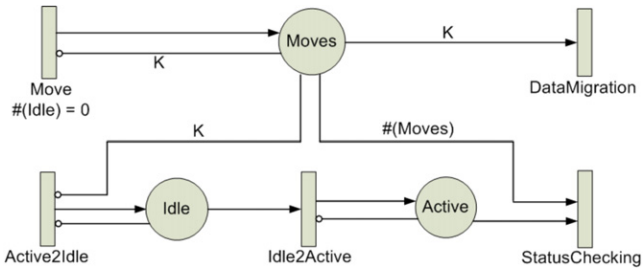



**Fig. 4.** The SPN model for DPM.

**Table 3**
The meanings of places and transitions defined in the SPN model for DPM.

| Symbol | Meaning |
|---|---|
| *Move* | A timed transition modeling MC movement |
| *Moves* | `Mark(Moves)` represents the number of movements |
| *DataMigration* | A timed transition modeling the migration of the data object between two data proxies |
| *Active2Idle* | A timed transition modeling the state transition of the MC from active mode to idle mode |
| *Idle* | `Mark(Idle)=1` means the MC is in idle mode |
| *Idle2Active* | A timed transition modeling the state transition of the MC from idle mode to active mode |
| *Active* | `Mark(Active)=1` means the MC is in active mode |
| *StatusChecking* | A timed transition modeling the event of checking the caching status after the MC reconnects |

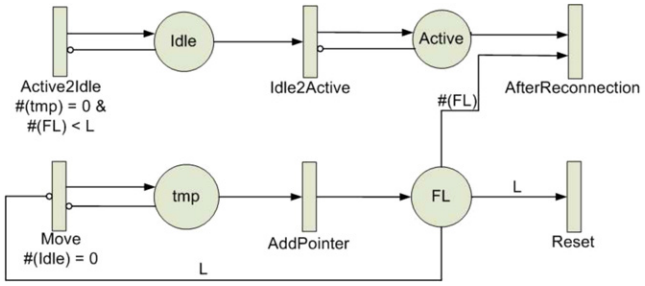



**Fig. 5.** The SPN model for CCM.

**Table 4**
The meanings of places and transitions defined in the SPN model for CCM.

| Symbol | Meaning |
|---|---|
| *Move* | A timed transition modeling MC movement |
| *tmp* | `Mark(tmp)=1` means that the MC just moves to a new MR |
| *AddPointer* | A timed transition modeling setting up a forwarding pointer between two neighboring MRs |
| *FL* | `Mark(FL)` returns the MC's current forwarding chain length |
| *Reset* | A timed transition modeling a location update event that resets the forwarding chain |
| *Active2Idle* | A timed transition modeling the state transition of the MC from active mode to idle mode |
| *Idle* | `Mark(Idle)=1` means the MC is in idle mode |
| *Idle2Active* | A timed transition modeling the state transition of the MC from idle mode to active mode |
| *Active* | `Mark(Active)=1` means the MC is in active mode |
| *AfterReconnection* | A timed transition modeling the event of retrieving IRs from the forwarding chain head received during the MC's disconnection and sending the gateway a location binding update message after the MC reconnects |

associated with a new MR, a token is put into place *Moves*, which represents the number of times the MC has moved since the most recent migration of the data object under consideration.

- When the number of movements since the last data object migration reaches $K$, i.e., the number of tokens in place *Moves* is accumulated to $K$, transition *DataMigration* is enabled and fired, representing the event of migrating the data object from the data proxy where it is currently cached to the one running on the new serving MR.
- An MC typically switches alternatively between active mode and idle mode during its stay in a WMN. Initially the MC is in active mode, and can send and receive packets. After staying in active mode for a period of time, the MC is switched to idle mode to save battery life. This is modeled by transition *Active2Idle*, the transition rate of which is $\omega_s$. After transition *Active2Idle* is fired, a token is put into place *Idle*, representing that the MC is switched to idle mode. The MC reconnects to the WMN after being in idle mode for some time. This is modeled by transition *Idle2Active*, the transition rate of which is $\omega_w$.
- When the MC reconnects, it sends a query message to the data proxy where the data object under consideration is cached to check its caching status. This event is modeled by transition *StatusChecking*. If the cached copy is still valid, it is migrated to the new data proxy running on the MC's current serving MR; otherwise, the data proxy simply responses with a message telling that the cached copy is obsolete. After the status checking, the number of movements since the last data migration is reset to zero, i.e., all tokens in place *Moves* are consumed by transition *StatusChecking*.

Fig. 5 depicts the SPN model for CCM. Table 4 defines the meanings of places and transitions in the model. The SPN model for CCM is constructed as follows:

- The event of MC movement is modeled by transition *Move*, the transition rate of which is $\sigma$. When an MC moves to and is associated with a new MR, a token is put into place *tmp*, which subsequently enables and fires transition *AddPointer*, meaning that a new forwarding pointer is setup between the old and new serving MRs. After transition *AddPointer* is fired, a token is put into place *FL* that represents the length of the forwarding chain, indicating that the forwarding chain length is increased by 1.
- When the length of the forwarding chain reaches $L$, i.e., the number of tokens in place *FL* is accumulated to $L$, transition *Reset* is enabled and fired, representing that the current forwarding chain is reset and the new current serving MR becomes the MC's new forwarding chain head.
- Transitions *Active2Idle* and *Idle2Active* represent the same physical meanings as in the SPN model for DPM.
- When the MC reconnects, it sends a query message to its current forwarding chain head to retrieve any IRs received by the forwarding chain head during its disconnection. The MC also sends a location binding update message to the gateway to update its location information, i.e., the address of the forwarding chain head. After the location update, the current serving MR becomes the MC's new forwarding chain head. Transition *AfterReconnection* models the above events performed by the MC after it reconnects.

### 5.2. Parameterization

Transition *DataMigration* in the SPN model for DPM represents the event of migrating a data object between two data proxies when the threshold $K$ with respect to the data object is reached. In this event, the MC that initiates the migration sends a data migration request to the data proxy where the data object is cached, asking it to migrate the data object to the data proxy on the MC's new serving MR. After the data migration is completed, the new serving MR sends a data migration acknowledgement to the MC. The signaling cost incurred by the event is $2K\tau + 2\tau$ because the distance between the two data proxies is $K$ hops. Therefore, the transition rate of *DataMigration* denoted by $\mu_{\text{DataMigration}}$ is calculated as:

$$\mu_{\text{DataMigration}} = \frac{1}{2K\tau + 2\tau}. \quad (1)$$

Transition *StatusChecking* in the SPN model for DPM represents the event of checking the caching status of a data object in a data proxy after the MC that initiates the status checking reconnects, and if the data object is still valid, migrating it to the data proxy on the MC's new serving MR. Therefore, the signaling cost for status checking upon reconnection is the sum of the cost for sending the status checking request and the cost for migrating the data object if it is still valid or for transmitting the IR if the data object has already been invalidated. The cost is $2\beta\tau + 2\tau$ because the distance between the current serving MR of the MC after it reconnects and its serving MR before disconnection is $\beta$ hops. The transition rate of *StatusChecking* denoted by $\mu_{\text{StatusChecking}}$ is therefore calculated as:

$$\mu_{\text{StatusChecking}} = \frac{1}{2\beta\tau + 2\tau}. \quad (2)$$

Transition *Reset* in the SPN model for CCM represents the event of resetting the current forwarding chain of an MC. In this event, the MC sends a location binding update message to the gateway to update its location information, i.e., the address of the new forwarding chain head. The gateway responses with a location binding update confirmation. The signaling cost incurred in this event is $2\alpha\tau + 2\tau$. Therefore, the transition rate denoted by $\mu_{\text{Reset}}$ can be derived as follows:

$$\mu_{\text{Reset}} = \frac{1}{2\alpha\tau + 2\tau}. \quad (3)$$

Transition *AfterReconnection* in the SPN model for CCM represents two events after an MC reconnects. Specifically, the MC sends a query message to its forwarding chain head before disconnection to retrieve any IRs received by the head during its reconnection. The MC also sends a location binding update message to the gateway to update its location information. The signaling cost incurred is $2(\alpha+\beta)\tau + 4\tau$. Therefore, the transition rate denoted by $\mu_{\text{AfterReconnection}}$ can be derived as follows:

$$\mu_{\text{AfterReconnection}} = \frac{1}{2(\alpha+\beta)\tau + 4\tau}. \quad (4)$$

### 5.3. Performance metric

We use the *total communication cost incurred per-time-unit (per-second) per-MC per-object* as the metric for performance evaluation. The total communication cost is measured as the number of hops because we normalize one-hop communication cost between two neighboring MRs, $\tau$, to 1. It is worth noting that since the performance metric used is on a per-MC, per-object, and per-time-unit basis, even a small performance gain of 5%–10% is considered significant over time, over all MCs, and over all data objects in the system. For DPM, the total communication cost ($C_{\text{DPM}}$) consists of the query cost ($C_{\text{query}}$), the invalidation cost ($C_{\text{invalidation}}$), the signaling cost for data migration ($C_{\text{migration}}$), and the signaling cost for status checking upon reconnection ($C_{\text{reconnection}}$). Using these cost terms, $C_{\text{DPM}}$ is calculated as follows:

$$C_{\text{DPM}} = \lambda' \cdot C_{\text{query}} + (\mu + \delta + \eta') \cdot C_{\text{invalidation}} + \sigma' \cdot C_{\text{migration}} + \omega \cdot C_{\text{reconnection}}. \quad (5)$$

In the above equation, $\lambda'$ and $\eta'$ denote the effective data query rate and the effective data update rate of the MC under consideration, respectively. $\sigma'$ denotes the effective mobility rate of the MC. These are effective rates because the MC cannot access or update any data object during its disconnection. These effective rates are calculated by:

$$\begin{aligned} \sigma' &= P_{\text{active}}\sigma \\ \lambda' &= P_{\text{active}}\lambda \\ \eta' &= P_{\text{active}}\eta. \end{aligned} \quad (6)$$

Here $P_{\text{active}}$ denotes the probability that the MC is in active mode, calculated by the ratio of the average active duration over the sum of the average active duration and the average idle duration, as follows:

$$P_{\text{active}} = \frac{\frac{1}{\omega_s}}{\frac{1}{\omega_s} + \frac{1}{\omega_w}} = \frac{\omega_w}{\omega_w + \omega_s}. \quad (7)$$

Below we parameterize various cost terms in Eq. (5). We exclude the communication cost for data object or IR transmission between the server and the gateway because it is not part of the overall cost incurred to the WMN. The query cost incurred by DPM in the case of a cache hit consists of the cost for sending the query to the data proxy, and the cost for delivering the queried data object to the MC. In the case of a cache miss, the query cost consists of the cost for sending the query to the data proxy, the cost for forwarding the query to the gateway, the cost for transmitting the data object from the gateway to the proxy, and finally the cost for delivering the queried data object to the MC. Therefore, $C_{\text{query}}$ is calculated as (with $n_m$ denoting the number of tokens in place *Moves*):

$$C_{\text{query}} = \begin{cases} 2n_m\tau + 2\tau & \text{if cache hit} \\ 2(\alpha + n_m)\tau + 2\tau & \text{if cache miss.} \end{cases} \quad (8)$$

The invalidation cost incurred by DPM in the case that the MC under consideration updates a cached data object consists of the cost for sending the updated data object and IR to the data proxy, and subsequently to the gateway, and the cost for the delivery of the invalidation acknowledgement. If the invalidation is due to updates from other MCs, the invalidation cost consists of the cost for pushing the IR from the gateway to the MC under consideration and the cost for transmitting the invalidation acknowledgement. Therefore, $C_{\text{invalidation}}$ is given by (with $\text{MC}_0$ representing the MC under consideration):

$$C_{\text{invalidation}} = \begin{cases} 2\alpha\tau & \text{if update by other MCs} \\ 2(n_m + \alpha)\tau + 2\tau & \text{if update by } \text{MC}_0 \text{ itself.} \end{cases} \quad (9)$$

The signaling cost for data migration in DPM is for transmitting the data migration request and acknowledgement, and for transmitting the data object between two data proxies that are $K$ hops away from each other. $C_{\text{migration}}$ is therefore calculated as:

$$C_{\text{migration}} = 2K\tau + 2\tau. \quad (10)$$

The signaling cost for status checking upon reconnection in DPM is the sum of the cost for sending the status checking request and the cost for migrating the data object if it is still valid or for transmitting the IR if the data object has already been invalidated. Because the distance between the current serving MR of the MC after it reconnects and its serving MR before disconnection is $\beta$ hops, $C_{\text{reconnection}}$ is calculated as:

$$C_{\text{reconnection}} = 2\beta\tau + 2\tau. \quad (11)$$

For CCM, the total communication cost ($C_{\text{CCM}}$) consists of the query cost ($C_{\text{query}}$), the invalidation cost ($C_{\text{invalidation}}$), the signaling cost for location management ($C_{\text{location}}$), and the signaling cost for cache status checking upon reconnection ($C_{\text{reconnection}}$). Using these cost terms, $C_{\text{CCM}}$ is calculated as:

$$C_{\text{CCM}} = \lambda' \cdot C_{\text{query}} + (\mu + \delta + \eta') \cdot C_{\text{invalidation}} + \sigma' \cdot C_{\text{location}} + \omega \cdot C_{\text{reconnection}}. \quad (12)$$

The query cost incurred by CCM in the case of a cache hit is zero because the data object is retrieved locally from the cache. In the case of a cache miss, the query cost consists of the cost for sending the query to the gateway and the cost for transmitting the data object from the gateway to the MC following the forwarding chain. Therefore, $C_{\text{query}}$ is calculated as (with $l_f$ denoting the current forwarding chain length):

$$C_{\text{query}} = \begin{cases} 0 & \text{if cache hit} \\ (\alpha + l_f)\tau + \alpha\tau + 2\tau & \text{if cache miss.} \end{cases} \quad (13)$$

The calculation of the invalidation cost in CCM depends on where the update to the invalidated data object originates. In the case that the update is from the server or other MCs, two situations can arise: (a) if the MC under consideration is in idle mode, the cost incurred is for transmitting the IR and invalidation acknowledgement between the gateway and the MC's forwarding chain head, and (b) if the MC under consideration is in active mode, the cost incurred is for transmitting the IR and invalidation

acknowledgement between the gateway and the MC. If the update is from the MC under consideration ($MC_0$), the invalidation cost consists of the cost for sending the IR and the updated data object to the gateway, and the cost for transmitting the invalidation acknowledgement. Therefore, $C_{\text{invalidation}}$ is calculated as:

$$C_{\text{invalidation}} = \begin{cases} 2\alpha\tau & \text{if update by other MCs or the server and } MC_0 \text{ is active} \\ (2\alpha + l_f)\tau + 2\tau & \text{if update by other MCs or the server and } MC_0 \text{ is idle} \\ (2\alpha + l_f)\tau + 2\tau & \text{if update by } MC_0 \text{ itself.} \end{cases} \quad (14)$$

The signaling cost for location management in CCM is the cost for setting up a forwarding pointer between two neighboring MRs, if the forwarding chain length after the movement is less than the threshold $L$, or the cost for location update if the forwarding chain length after the movement reaches the threshold $L$. In the latter case, a location binding update message is sent to the gateway in this case to update the location information of the MC, i.e., the address of its forwarding chain head. Therefore, $C_{\text{location}}$ is calculated as:

$$C_{\text{location}} = \begin{cases} 4\tau & \text{if } f_l < L \\ 2\alpha\tau + 2\tau & \text{if } f_l = L. \end{cases} \quad (15)$$

The signaling cost for status checking upon reconnection in CCM consists of the cost for transmitting the status checking request and the IR if there are cached data object that have been invalidated during the disconnection, and the cost for location update. Therefore, $C_{\text{reconnection}}$ is calculated as:

$$C_{\text{reconnection}} = 2(\alpha + \beta)\tau + 4\tau. \quad (16)$$

A query for a data object results in a cache hit if there is no update to the object between this query and the most recent query in the past. There will be a cache miss if there is at least one update to the object in between two consecutive queries. Therefore, the cache hit ratio of a data object denoted by $P_{\text{hit}}$ can be calculated by the average number of successive accesses that can be done during the interval between two consecutive updates, as follows:

$$P_{\text{hit}} = \frac{\lambda'}{\lambda' + \mu + \delta + \eta'}. \quad (17)$$

An MC may switch between active mode and idle mode during its stay in a WMN. The average interval between two consecutive reconnections of the MC is the sum of the average active duration and the average idle duration, i.e., $\frac{1}{\omega_w} + \frac{1}{\omega_s}$. The rate of reconnection denoted by $\omega$ is therefore calculated as follows:

$$\omega = \frac{1}{\frac{1}{\omega_w} + \frac{1}{\omega_s}} = \frac{\omega_w \times \omega_s}{\omega_w + \omega_s}. \quad (18)$$

## 6. Performance analysis and numerical results

In this section, we analyze the performance of APPCCM, in terms of the total communication cost incurred per-time-unit (per-second) per-MC per-object. We also carry out a comparative performance study to compare APPCCM with non-adaptive cache consistency management schemes that always cache a data object at the MC, or at the MC's current serving MR for supporting mobile data access in WMNs. The numerical results are obtained by first defining a SPN model as shown in Fig. 4 or Fig. 5 using SPNP [7], and then evaluating the SPN model by assigning rewards to states of the system based on the equations presented above. We define a parameter called *query to mobility ratio* (QMR) indicating how often an MC performs data access relative to its mobility speed. The QMR of an MC with respect to a data object is defined as: QMR $= \frac{\lambda}{\sigma}$.

**Table 5**
Parameters and their values used in performance analysis.

| Parameter | Value |
|---|---|
| QUR | {0.125, 0.25, 0.5, 1.0, 2.0, 4.0, 8.0} |
| QMR | {1, 2, 4, 8, 16, 32, 64, 128, 256} |
| $P_{\text{active}}$ | $\{\frac{1}{9}, \frac{1}{5}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{4}{5}, \frac{8}{9}\}$ |
| $\omega_w$ | $\frac{1}{600}$ |
| $\omega_s$ | $\frac{1}{1200}$ |
| $\alpha$ | {10, 20, 30} |
| $\beta$ | {5, 10, 20} |
| $\tau$ | 1 |

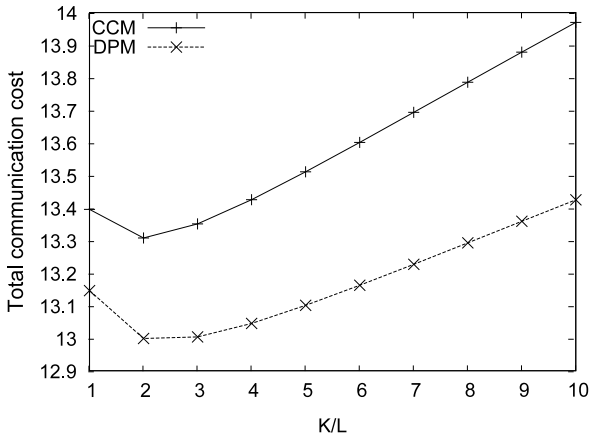



**Fig. 6.** Cost vs. $K/L$ (QUR = 0.5; QMR = 8.0).

Table 5 lists the parameters and their default values used in the performance evaluation. The time unit used in the paper is in seconds. We select a wide range of values for QMR, QUR, and $P_{\text{active}}$ to cover diverse data query/update, mobility and service patterns and to test their effects on performance characteristics of APPCCM. The physical meaning of $\omega_w$ is the reciprocal of the average duration of disconnection of an MC before reconnection with the WMN. Therefore, $\omega_w = \frac{1}{600}$ means that the MC stays disconnected for a duration of 10 min on average before switching to active mode and reconnecting with the WMN. Similarly, $\omega_s = \frac{1}{1200}$ means that the MC stays in active mode for a duration of 20 min on average before disconnecting from the WMN and changing to idle mode. The values of $\alpha$ and $\beta$ are chosen to model WMNs of different dimensions, using hop counts to measure distances. For example, we vary the distance between the gateway and an arbitrary MR, i.e., $\alpha$ from 10 to 30 to simulate WMNs of different sizes. All costs presented below are normalized with respect to $\tau = 1$.

### 6.1. Performance evaluation of APPCCM

Figs. 6 and 7 plot $C_{\text{DPM}}$ and $C_{\text{CCM}}$ as a function of $K/L$, under two different values of QUR. As both figures illustrate, for either mode, there exists an optimal threshold $K/L$ that minimizes the total communication cost, given a set of parameters characterizing the mobility and data query/update characteristics (with respect to the accessed data object) of the MC. DPM performs consistently better than CCM, when QUR = 0.5, whereas CCM is superior to DPM when QUR = 2.

Intuitively, DPM is expected to perform better than CCM under circumstances when the access rate is lower than the update rate, because DPM reduces invalidation costs at the expense of increased query costs. In contrast, CCM is expected to be superior to DPM under other circumstances when the access rate is higher than the update rate, because CCM incurs smaller query costs than DPM under the same cache hit ratio, at the expense of increased invalidation costs.

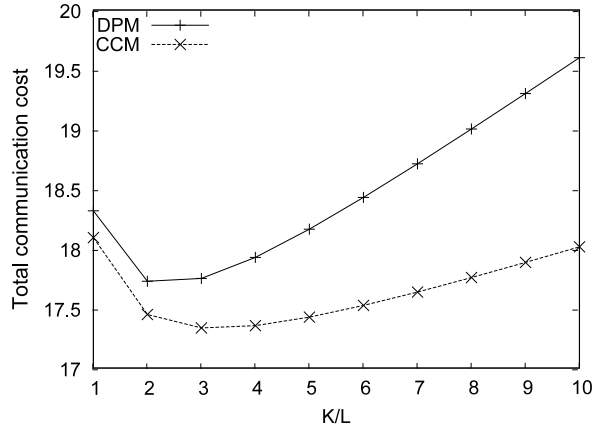



**Fig. 7.** Cost vs. $K/L$ (QUR = 2.0; QMR = 8.0).

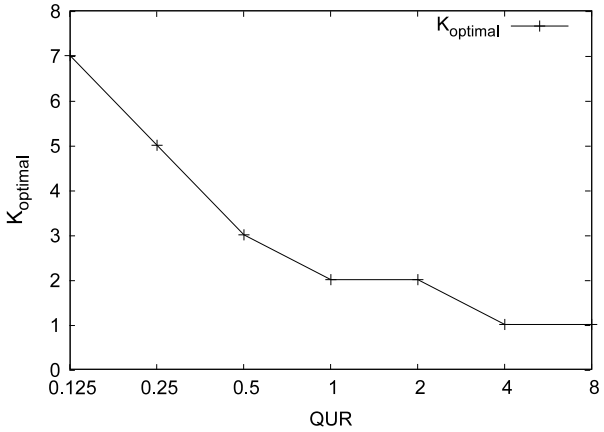



**Fig. 9.** The optimal threshold $K_{optimal}$ vs. QUR in DPM.

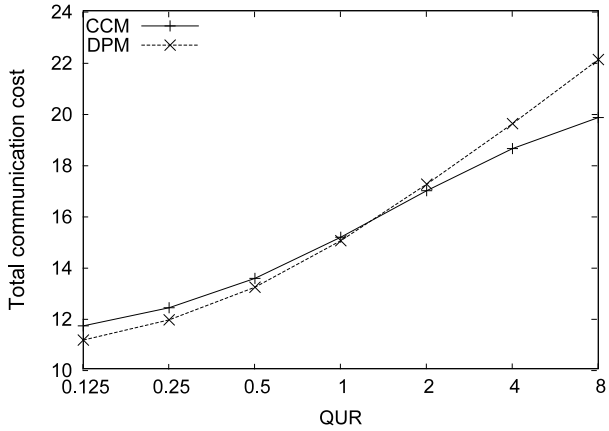



**Fig. 8.** Cost vs. QUR.

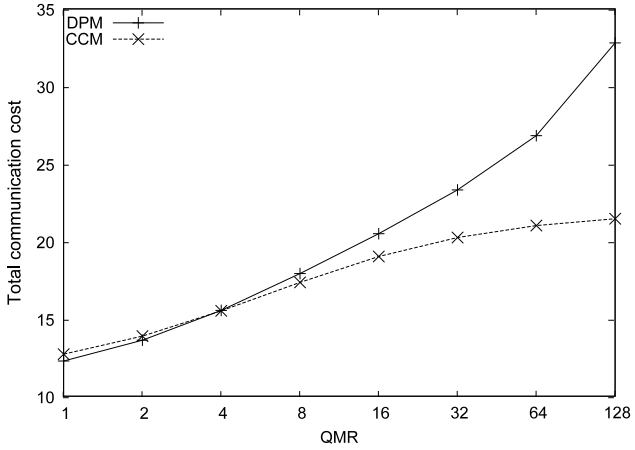



**Fig. 10.** Cost vs. QMR.

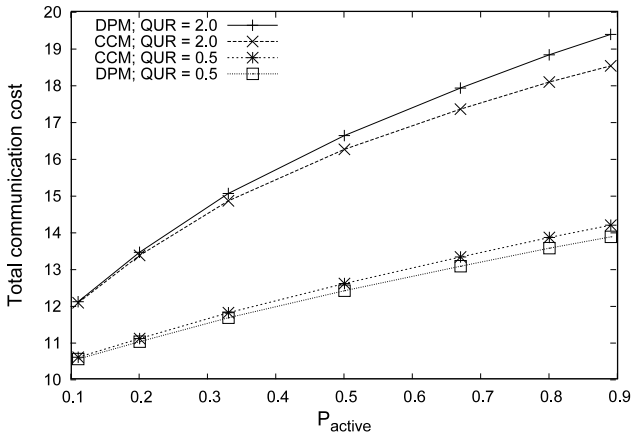



**Fig. 11.** Cost vs. $P_{active}$ under two different values of QUR.

The effect of tradeoff on the performance of DPM and CCM is clearly illustrated by Fig. 8, which compares $C_{DPM}$ and $C_{CCM}$, as a function of QUR. As can be seen in the figure, initially when QUR is small, DPM performs better than CCM. As QUR increases, the performance gap between DPM and CCM decreases, and there exists a crossover point of QUR beyond which CCM becomes superior to DPM. As illustrated by Fig. 8, when QUR is high, APPCCM behaves like CCM for which there is no data object migration, since the MC will store data objects locally in its own cache. When QUR is low, APPCCM behaves like DPM for which $K_{optimal}$ increases as the user mobility rate increases. Therefore, APPCCM will reduce the object migration overhead for highly mobile MCs, essentially avoiding inopportune data object migration among MRs when MCs have a high level of mobility.

Fig. 9 shows the optimal threshold $K_{optimal}$ in DPM as a function of QUR. As shown in both figures, the optimal threshold $K_{optimal}$ in DPM decreases with increasing QUR. This is because as QUR increases, the contribution of the query cost to the total communication cost becomes more significant, and therefore, a smaller $K_{optimal}$ is favored to reduce the query cost and consequently minimize the total communication cost.

Fig. 10 compares $C_{DPM}$ and $C_{CCM}$ as a function of QMR. We observe that this figure shows the same trend as in Fig. 8. There exists a crossover point of QMR beyond which CCM outperforms DPM. This is because with a fixed mobility rate and fixed update rates ($\mu$, $\delta$, and $\eta$), the access rate increases as QMR increases, and consequently QUR increases, resulting in the same trend as in Fig. 8.

Fig. 11 compares $C_{DPM}$ and $C_{CCM}$ as a function of $P_{active}$, under two different values of QUR. As the figure shows, both $C_{DPM}$ and $C_{CCM}$ increase monotonically with increasing $P_{active}$. This is because the volume of activities of an MC and consequently the communication cost it incurs increase as the MC spends increasingly more time in active mode than in idle mode. The figure also shows that DPM outperforms CCM when QUR = 0.5, whereas CCM is superior to DPM when QUR = 2.0. These results are well correlated with the trends exhibited in Figs. 6 and 7.

Fig. 12 investigates the sensitivity of performance evaluation to $\alpha$ and $\beta$. Specifically, Fig. 12 compares $C_{DPM}$ and $C_{CCM}$ as a function of QUR, under different combinations of $\alpha$ and $\beta$. Comparing the figure with Fig. 8, it can be observed that both figures exhibit very

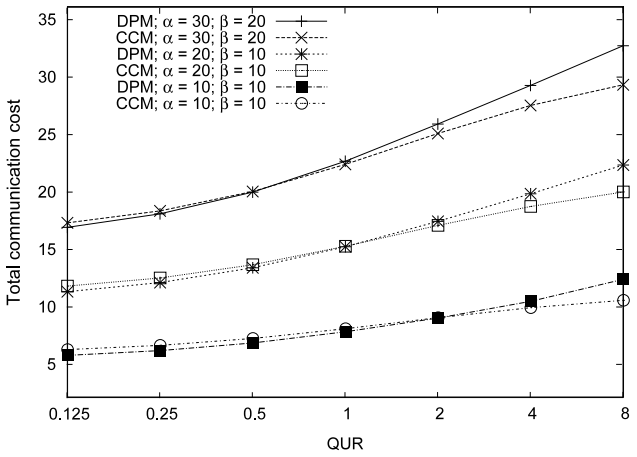



**Fig. 12.** Cost vs. QUR under different combinations of $\alpha$ and $\beta$.

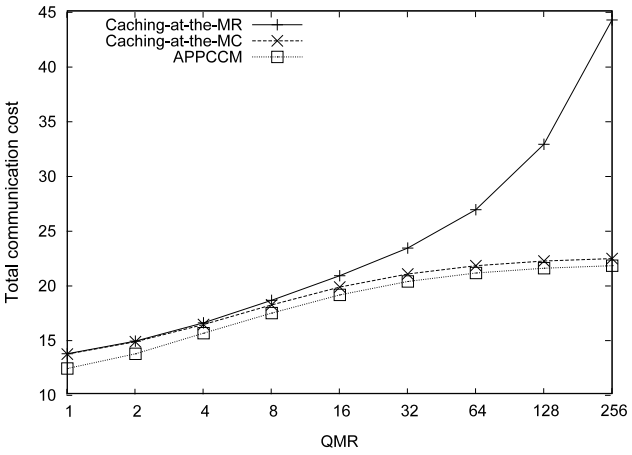



**Fig. 14.** Cost vs. QMR.

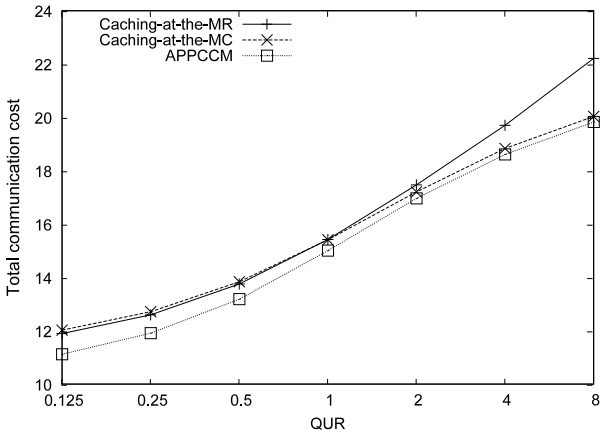



**Fig. 13.** Performance comparison: cost vs. QUR.

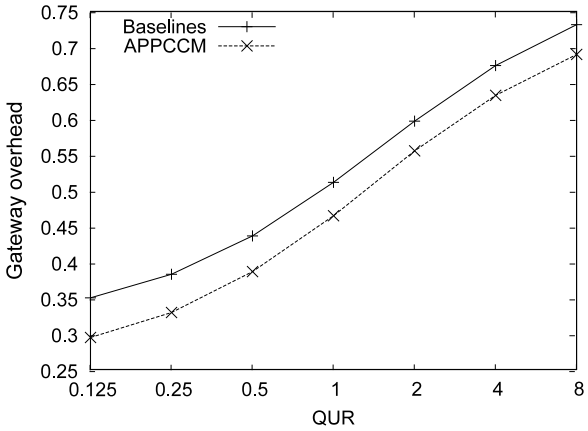



**Fig. 15.** Gateway overhead vs. QUR.

similar trends, regardless of the values of $\alpha$ and $\beta$. Here we again observe that there exists a crossover point of QUR beyond which CCM outperforms DPM, and the performance gain of CCM over DPM increases with increasing QUR. This figure also illustrates that APPCCM achieves good scalability in terms of different network sizes as measured by different values of $\alpha$ and $\beta$. APPCCM achieves scalability essentially through network cost minimization, thus allowing more MCs to be accommodated by a WMN. The cost saving or the amount of network traffic reduced is cumulative and is more significant as the number of MCs increases.

### 6.2. Comparative performance study

In this section, we compare APPCCM with two non-adaptive cache consistency management schemes for supporting mobile data access. The first non-adaptive scheme always caches a data object at the MC directly (we call it caching-at-the-MC scheme). Therefore, it is essentially equivalent to existing asynchronous stateful-based schemes [2,21] augmented with optimal per-user pointer forwarding for integrated cache consistency and mobility management such that MCs can perform data access, data caching and cache consistency management while roaming and changing their locations in a WMN. The second non-adaptive scheme always caches a data object at the MC's current serving MR (we call it caching-at-the-MR scheme). Therefore, a data object cached by an MC at its current serving MR will be migrated to the new serving MR every time the MC moves and changes its location.

Figs. 13 and 14 compare the total communication cost between APPCCM and the two non-adaptive schemes, as a function of QUR and QMR, respectively. We see that APPCCM outperforms both non-adaptive schemes. More specifically, APPCCM performs significantly better than the caching-at-the-MC scheme initially when QUR/QMR is small. As QUR/QMR increases, the performance gap narrows, and at some point, APPCCM degenerates to the caching-at-the-MC scheme, or equivalently, CCM. This is because APPCCM adaptively uses CCM to cache a data object when QUR/QMR is relatively large. Therefore, the advantage of APPCCM over the caching-at-the-MC scheme is most pronounced when QUR/QMR is relatively small.

Below we explain the reason that APPCCM is always superior to the caching-at-the-MR scheme. When QUR/QMR is small, APPCCM adaptively uses DPM to cache a data object. Since APPCCM dynamically determines $K_{\text{optimal}}$ under which the total communication cost is minimized, APPCCM performs significantly better than the caching-at-the-MR scheme, when QUR/QMR is small. As QUR/QMR increases, APPCCM adaptively switches to CCM, because CCM is superior to DPM when QUR/QMR is large. Therefore, APPCCM is also superior to the caching-at-the-MR scheme when QUR/QMR is large. Here again we emphasize that since the performance metric used in this paper is on a per-MC, per-object and per-time-unit basis, so even a small performance gain of 5%–10% would be significant over time, over all MCs, and over all data objects in the system. These results demonstrate the advantage of APPCCM over non-adaptive schemes due to its ability to choose either DPM or CCM that can best balance the query cost and the invalidation cost and thus minimize the overall cost.

Figs. 15 and 16 compare the gateway overhead between APPCCM and the baseline schemes, as a function of QUR and QMR, respectively. The gateway overhead is measured by the traffic

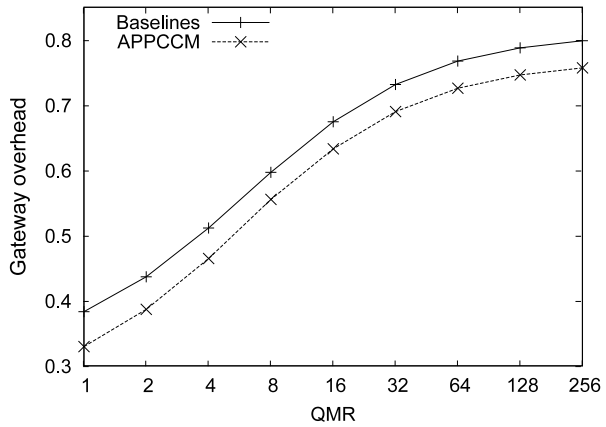



**Fig. 16.** Gateway overhead vs. QMR.

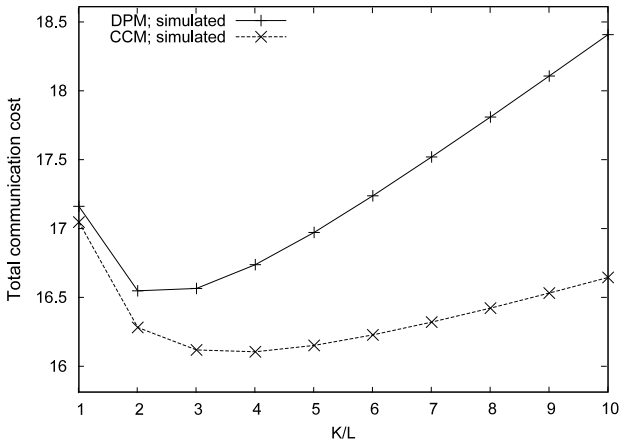



**Fig. 18.** Simulation validation: cost vs. $K/L$ (QUR = 2.0).

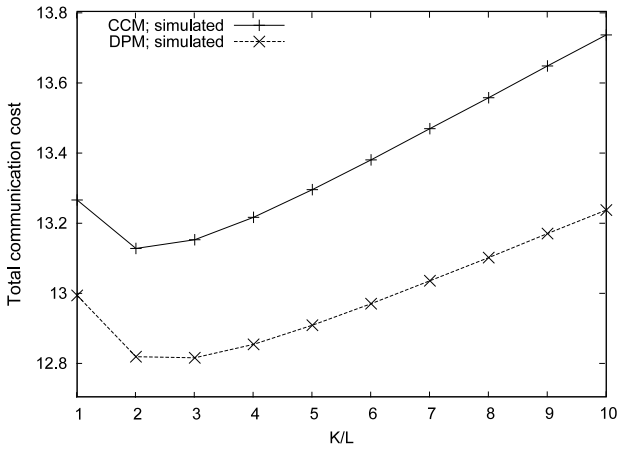



**Fig. 17.** Simulation validation: cost vs. $K/L$ (QUR = 0.5).

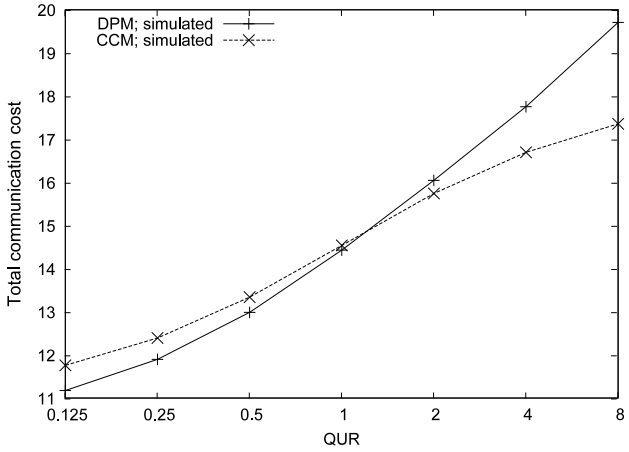



**Fig. 19.** Simulation validation: cost vs. QUR.

incurred at the gateway, i.e., the number of messages processed by the gateway per second. Messages considered include those used for carrying queries, data objects, IRs, and location/data proxy updates. As can be seen in the figures, APPCCM reduces the traffic at the gateway for approximately 5% to 10% on a per-second basis, compared with the two baseline schemes which incur the same amount of traffic at the gateway. In particular, we demonstrate that in case of a great number of data updates (when QUR is small reflecting a high data update rate) or a great number of location updates (when QMR is small reflecting a high mobility rate), APPCCM greatly reduces the gateway overhead compared with the two baseline schemes.

### 6.3. Simulation validation

We conduct extensive simulation to validate the analytical results obtained above, using a discrete simulation language called Simulation Model Programming Language (SMPL) [17]. In this simulation system, all operations in APPCCM are associated with discrete events, each with a state-dependent operation cost. For example, query processing operations, cache invalidation operations, data migration operations, and location update operations, are associated with events. Events are scheduled and executed in FIFO order, according to the algorithm description presented in Section 4. The average total communication cost is evaluated and reported. To ensure the statistical significance of simulation results, we use a batch mean analysis technique. Each simulation batch consists of a large number of runs and therefore a large number of observations for computing one batch average. The simulation runs for a minimum of 10 batches, and stops until the mean of the batch means collected is within 5% from the true mean with a confidence level of 95%. In the simulation study we use the same set of parameter values as those listed in Table 5.

Figs. 17 and 18 show the simulation results of $C_{DPM}$ and $C_{CCM}$ as a function of $K/L$, under two different values of QUR. Comparing both figures with Figs. 6 and 7, we observe that the simulation results are well-correlated with the analytical results, as demonstrated by the trends exhibited in the figures. This justifies that the analytical results are valid and there exists optimal thresholds $K_{optimal}$ and $L_{optimal}$, respectively, under which $C_{DPM}$ and $C_{CCM}$ are minimized.

Fig. 19 illustrates the simulation results of $C_{DPM}$ and $C_{CCM}$ as a function of QUR. Again, the simulation results show excellent correlation with the analytical results presented in Fig. 8. Similarly, the simulation results shown in Figs. 20 and 21 for the performance comparison between APPCCM and the two non-adaptive schemes also correlate well with the analytical results shown in Figs. 13 and 14.

## 7. Conclusion and applicability

In this paper, we proposed an adaptive per-user per-object cache consistency management scheme for mobile data access in WMNs, namely APPCCM, with the objective to improve data access performance as well as to mitigate the performance bottleneck at the gateways in WMNs. APPCCM supports strong data consistency

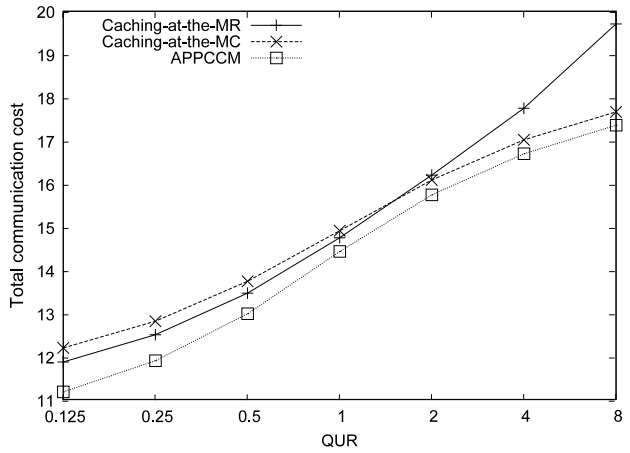



**Fig. 20.** Simulation validation: performance comparison vs. QUR.

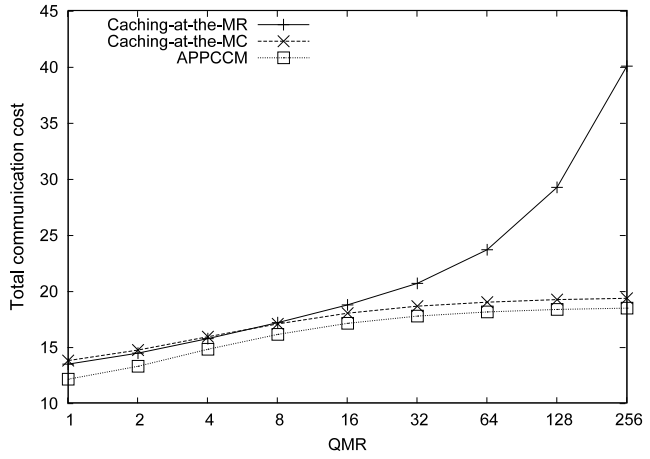



**Fig. 21.** Simulation validation: performance comparison vs. QMR.

semantics through integrated cache consistency and mobility management. APPCCM is adaptive, per-user and per-object, as one of two caching modes provided by APPCCM, namely DPM and CCM, is adaptively selected based on the MC's mobility and data query/update characteristics as well as operational and networking conditions of the WMN. We demonstrated via model-based analysis that APPCCM outperforms two non-adaptive cache consistency management schemes that always cache a data object at the MC or at the MC's current serving MR. The advantage of APPCCM is due to effective exploitation of the tradeoff between the query cost and invalidation cost realized by adaptively selecting the best cache consistency management strategy out of DPM and CCM.

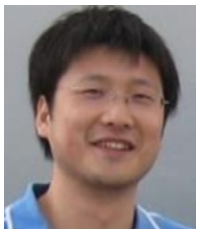
There is a variety of potential applications for which APPCCM is applicable. For example, digital news and magazine applications running on smartphones and mobile tablet computers such as iPhone and iPad, are typical applications demanding mobile data access. It is beneficial to use APPCCM for such applications in a WMN environment to provide better response time, reduce the Internet traffic flow billed to the users, and minimize the total communication cost incurred for maximizing the network's throughput.

To implement APPCCM on real mobile devices, the devices should have adequate computing power to perform the computational procedure presented in the paper and storage to store cached data objects. For mobile devices that are less powerful in computation, a table-lookup approach can be used to implement APPCCM without having to execute the computational procedure at runtime. Specifically, the decision regarding where to cache a data object and the optimal $K/L$ can be statically determined at the design time over a wide range of mobility and service characteristics and stored in a table for fast lookup. Then, during the execution of APPCCM, a simple table lookup can quickly determine the caching decision and the optimal $K/L$, when given a set of parameter values specifying the mobility and data query/update characteristics of the user.

In the future, we plan to investigate community-based sharing of data objects cached in a data proxy running on a mesh router concurrently serving multiple mobile users. We also plan to investigate concurrency control between the client and the server to support disconnected write operations. Another direction is to devise efficient integrated cache invalidation and replacement management that can further reduce the data access latency and the overall network cost.

## References

[1] I.F. Akyildiz, X. Wang, W. Wang, Wireless mesh networks: a survey, Computer Networks 47 (4) (2005) 445–487.
[2] D. Barbara, T. Imielinski, Sleepers and workaholics: caching strategies in mobile environments (extended version), The VLDB Journal 4 (4) (1995) 567–602.
[3] J. Cai, K.L. Tan, Energy efficient selective cache invalidation, Wireless Networks 5 (6) (1999) 489–502.
[4] G. Cao, A scalable low-latency cache invalidation strategy for mobile environments, IEEE Transactions on Knowledge and Data Engineering 15 (5) (2003) 1251–1265.
[5] I.R. Chen, T.M. Chen, C. Lee, Performance evaluation of forwarding strategies for location management in mobile networks, The Computer Journal 41 (4) (1998) 243–253.
[6] S.M. Das, H. Pucha, Y.C. Hu, Mitigating the gateway bottleneck via transparent cooperative caching in wireless mesh networks, Ad Hoc Networks 5 (6) (2007) 680–703.
[7] C. Hirel, B. Tuffin, K.S. Trivedi, Spnp: Stochastic petri nets, Version 6.0, in: 11th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools, Schaumburg, Illinois, United States, 2000, pp. 354–357.
[8] Q. Hu, D.K. Lee, Cache algorithms based on adaptive invalidation reports for mobile environments, Cluster Computing 1 (1) (1998) 39–50.
[9] T. Imielinski, B.R. Badrinath, Data management for mobile computing, SIGMOD Record 22 (1) (1993) 34–39.
[10] J. Jing, A. Elmagarmid, A.S. Helal, Bit-sequences: an adaptive cache invalidation method in mobile client/server environments, Mobile Networks and Applications 2 (2) (1997) 115–127.
[11] J. Jing, A. Helal, A. Elmagarmid, Client–server computing in mobile environments, ACM Computing Surveys 31 (2) (1999) 117–157.
[12] A. Kahol, S. Khurana, S.K.S. Gupta, P.K. Srimani, A strategy to manage cache consistency in a disconnected distributed environment, IEEE Transactions on Parallel and Distributed Systems 12 (7) (2001) 686–700.
[13] J.J. Kistler, M. Satyanarayanan, Disconnected operation in the Coda file system, ACM Transactions on Computer Systems 10 (1) (1992) 3–25.
[14] Y. Li, I.R. Chen, APPCCM: adaptive per-user per-object cache consistency management for mobile client–server applications in wireless mesh networks, in: 35th IEEE Conference on Local Computer Networks, Denver, Colorado, United States, 2010, pp. 128–135.
[15] Y. Li, I.R. Chen, Design and performance analysis of mobility management schemes based on pointer forwarding for wireless mesh networks, IEEE Transactions on Mobile Computing 10 (3) (2011) 349–361.
[16] Y.B. Lin, W.R. Lai, J.J. Chen, Effects of cache machanism on wireless data access, IEEE Transactions on Wireless Communications 2 (6) (2003) 1247–1258.
[17] M.H. MacDougall, Simulating Computer Systems, MIT Press, 1987.
[18] A. Madhukar, T. Ozyer, R. Alhajj, Dynamic cache invalidation scheme for wireless mobile environments, Wireless Networks 15 (6) (2009) 727–740.
[19] N. Nandiraju, L. Santhanam, B. He, J. Wang, D. Agrawal, Wireless mesh networks: current challenges and future directions of web-in-the-sky, IEEE Wireless Communications 14 (4) (2007) 79–89.
[20] M. Satyanarayanan, Fundamental challenges in mobile computing, in: 15th Annual ACM Symposium on Principles of Distributed Computing, Philadelphia, Pennsylvania, United States, 1996, pp. 1–7.
[21] K.L. Tan, J. Cai, B.C. Ooi, An evaluation of cache invalidation strategies in wireless environments, IEEE Transactions on Parallel and Distributed Systems 12 (8) (2001) 789–807.
[22] Z. Wang, M. Kumar, S.K. Das, H. Shen, Dynamic cache consistency schemes for wireless cellular networks, IEEE Transactions on Wireless Communications 5 (2) (2006) 366–376.
[23] Y. Xiao, H. Chen, Optimal callback with two-level adaptation for wireless data access, IEEE Transactions on Mobile Computing 5 (8) (2006) 1087–1102.
[24] J. Yin, L. Alvisi, M. Dahlin, C. Lin, Volume leases for consistency in large-scale systems, IEEE Transactions on Knowledge and Data Engineering 11 (4) (1999) 563–576.



**Yinan Li** received the B.S. degree in Computer Science from Xi'an Jiaotong University in China, and the M.S. degree in Computer Science from the University of Tennessee in 2008. He is currently a Ph.D. student in the Department of Computer Science at Virginia Tech. His research interests include wireless networks, mobile ad hoc networks, sensor networks, network security, high performance computing, and dependable computing.

**Ing-Ray Chen** received the B.S. degree from the National Taiwan University, Taipei, Taiwan, and the MS and Ph.D. degrees in computer science from the University of Houston. He is a professor in the Department of Computer Science at Virginia Tech. His research interests include mobile computing, wireless networks, security, multimedia, real-time intelligent systems, and reliability and performance analysis. Dr. Chen currently serves as an editor for The Computer Journal, Wireless Personal Communications, Wireless Communications and Mobile Computing, Security and Communication Networks, and International Journal on Artificial Intelligence Tools. He is a member of the IEEE/CS and ACM.