APPCCM: Adaptive Per-User Per-Object Cache Consistency Management for Mobile Client-Server Applications in Wireless Mesh Networks

Yinan Li, Ing-Ray Chen
Department of Computer Science
Virginia Polytechnic Institute and State University
Email: {yinan926, irchen}@vt.edu

***Abstract*—In this paper, we propose and analyze APPCCM: an adaptive per-user per-object cache consistency management scheme for mobile Internet client-server applications in wireless mesh networks (WMNs). The objective of the proposed scheme is two-fold: to speedup data access and to mitigate the performance bottleneck at WMN gateways. In our proposed adaptive scheme, a data object can be cached at the mesh client (MC) directly or at a mesh router (MR) dynamically selected by APPCCM, such that the overall network cost incurred for the WMN to process the MC's mobility and data access/update events is minimized. APPCCM is adaptive, per-user and per-object as the decision of where to cache a data object accessed by an MC depends on the MC's mobility and object access/update characteristics, and the WMN conditions. We demonstrate via model-based analysis that APPCCM outperforms non-adaptive schemes that always cache a data object at the MC, or at the MC's current serving MR for cache consistency management in WMNs.**

I. Introduction

Wireless Mesh Networks (WMNs) are emerging in recent years as a promising standard for next-generation broadband wireless networks and a cost-effective solution for last-mile broadband wireless Internet access. A WMN consists of two types of components: wireless mesh routers (MRs) and mesh clients (MCs) [1]. MRs form a static *wireless mesh backbone* for MCs. Network traffic is relayed within the wireless mesh backbone in a multi-hop manner. A WMN is seamlessly interconnected to the Internet through the gateway functionality of MRs. MCs are (portable) devices that have wireless access capability, e.g., laptops, netbooks, PDAs, cell phones, etc.

In this paper, we investigate data caching and cache consistency management for mobile Internet client-server applications in WMNs. Besides the benefit of reducing the average access latency, data caching is particularly beneficial for such applications in WMNs for a major reason. Specifically, because WMNs are proposed to be a cost-effective solution for last-mile broadband Internet access, Internet traffic, which always passes through the gateway, is expected to dominate network traffic in WMNs. The implication is that the gateway is potentially the bottleneck under conditions of heavy Internet traffic load. Caching is an efficient solution for mitigating the problem because it can significantly reduce the overhead of messages passing through the gateway in a mobile Internet client-server application [2].

We propose an adaptive per-user per-object cache consistency management scheme named APPCCM for mobile Internet client-server applications in WMNs. APPCCM is adaptive as it adaptively makes decision of where to cache a data object such that the total network traffic is minimized. APPCCM provides two caching mechanisms: a data object can be cached either directly by the MC, or at a *data proxy* running on an MR dynamically selected by APPCCM. We use the terms *client-cache mechanism* (CCM) and *data-proxy mechanism* (DPM) to represent the two mechanisms throughout the paper. A data proxy is essentially a cache maintained by an MR, similar to a *ProxyCache* in [3]. The rationale of using data proxies is that it is beneficial to cache a data object at a data proxy rather than at the MC under certain circumstances.

APPCCM is on a per-user per-object basis because for each individual MC, the decision of where to cache accessed data objects is made independently for each object based on the MC's mobility and object access/update characteristics, and the WMN conditions. For each of the two caching mechanisms, we develop a stochastic Petri net (SPN) model to evaluate the overall communication cost incurred under the mechanism. By comparing the overall communication cost incurred by the two mechanisms, we can adaptively choose the one with a smaller cost. Particularly for DPM, APPCCM determines *when* a cached object should be migrated between two data proxies due to MC mobility. To model the optimal time point at which the cached object should be migrated such that the overall communication cost is minimized, we use a threshold denoted by K of the distance between the MC's serving MR and the data proxy where the data object is cached.

The rest of the paper is organized as follows. Section II surveys existing work and contrasts our scheme with existing schemes. The proposed APPCCM scheme is presented in Section III. In Section IV, we develop analytical models based on stochastic Petri net for evaluating APPCCM. Performance analysis and comparison are carried out in Section V. The paper concludes with Section VI.

II. 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 as proposed by Barbara and Imielinski [4] 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. Mobile 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 [5], [6], [7], [8], [9], [10], [11], [12]. These schemes fall into two categories: *stateful-based* schemes and *stateless-based* schemes [13].

In stateful-based schemes, the server is stateful as it keeps information about where a data object is cached. Whenever the server updates a data object, it asynchronously sends an IR to those mobile hosts that keep a cached copy of the updated data object. In stateless-based approaches, the server either synchronously or asynchronously broadcasts an IR when a data object is updated. Most of the existing schemes basically deal with three issues: the content of IRs, the ways in which invalidation is performed, and the support the server needs to provide [13]. None of these schemes is designed for WMNs, and therefore cannot be applied directly for cache invalidation in WMNs without considerable modification and performance penalty. Additionally, one important limitation of existing schemes is that they generally do not consider the effect of host mobility. In contrast, APPCCM is stateful-based and specifically designed for WMNs, with consideration of the characteristics of WMNs. For example, the existence of the wireless mesh backbone consisting of MRs that have minimum mobility and good processing capability and expandable memory capacity (via USB-based flash or hard drives) makes it possible to perform data caching on the MRs. Moreover, APPCCM addresses integrated cache consistency management and MC mobility support. Another limitation of existing schemes is that mobile hosts are assumed to have read-only permissions. In APPCCM, we consider that MCs can both query and update any data objects.

III. The Proposed APPCCM Scheme

In this paper, we consider a scenario in which MCs within a WMN access data objects on a server that is located outside of the WMN and 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.

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. 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 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 any MCs within the WMN. Therefore, the gateway is like a replica of the server.

A. DPM versus CCM

As introduced in Section I, there are two caching mechanisms 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. Currently available MRs have sufficient computing power and storage capabilities to perform both routing and data caching [2]. 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.

Specifically, when a data object is updated more frequently than being accessed by an MC such that the cost of invalidation is dominating, it may be beneficial to cache the data object at a data proxy rather than locally at the MC to reduce the cost of invalidation 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 cost of access 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 cost of accessing a data object and the cost of invalidating a cached copy of the data object. APPCCM exploits this tradeoff and adaptively decides on a per-user per-object basis where to cache a data object.

B. Data Access and Caching

TABLE I
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	Indicating 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 when the cached copy of the data object is most recently updated due to query

In APPCCM, each MC maintains a *caching status table* that keeps the caching status of each data object it has accessed. The caching status table has four fields as shown and explained in Table I.

When an MC receives a 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. Depending on the result of this table lookup, the query is answered accordingly in different ways. The pseudo code given 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 update data object to the server and the IR to those MCs and data proxies that keep a cached copy of the data object being invalidated.

Fig. 1 illustrates query and update processing in APPCCM. As the figure shows, MC1 attached to MR1 employs MR2 as a data proxy. 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 and the IR to those MCs and data proxies that keep a cached copy of the data object being invalidated.

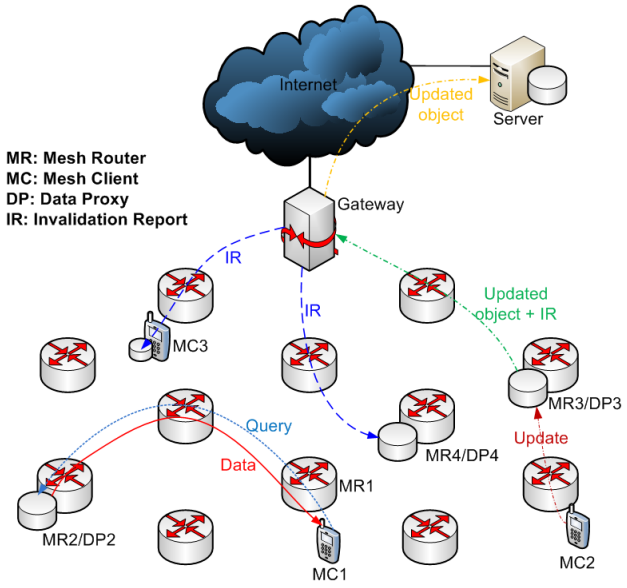

Fig. 1. Examples of query and update processing in APPCCM.

C. 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 mechanisms of APPCCM differently. Specifically, in DPM, an optimal threshold $K_{optimal}$ is derived 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 $K_{optimal}$ due to MC mobility, the data object is migrated to the data proxy on the MC's current serving MR. The optimal threshold $K_{optimal}$ that results in minimized overall communication cost is determined dynamically on a per-user per-object basis. Below we use K and $K_{optimal}$ to denote the threshold and the optimal threshold that minimizes the overall communication cost, respectively.

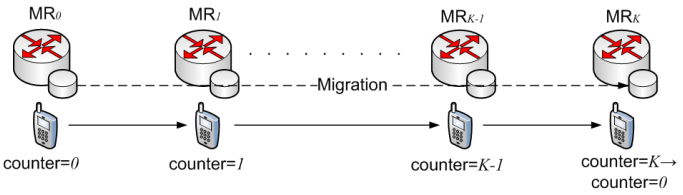

Fig. 2. Threshold-based data object migration in DPM.

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 movements of the MC since the data object's most recent migration. Each time when the MC moves to a new 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 CCM, a location management scheme is employed to track MC locations in order for the gateway to deliver IRs to the destination MCs. We develop a per-user location management scheme for CCM based on per-user pointer forwarding for WMNs [14] . In this scheme, the gateway maintains a location database where the address of the forwarding chain head of each MC is kept. An optimal threshold of the forwarding chain length denoted as L under which the overall communication cost of 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 point of attachment to a new 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.

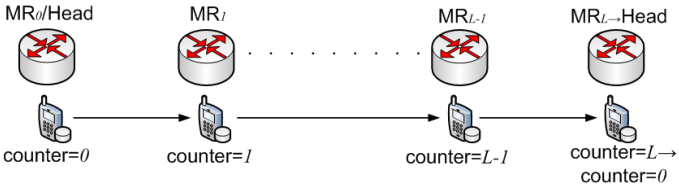

Fig. 3. The location management scheme in CCM.

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 when 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, and the counter is reset to zero.

IV. Analytical Modeling

In this section, we develop analytical models based on stochastic Petri net (SPN) for DPM and CCM. Table II presents parameters and notations used in performance modeling and analysis.

A. 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 III. `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 σ. When an MC moves to and is 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 its current data proxy to the data proxy on the new current serving MR.

TABLE II
Parameters and notations used in performance modeling.

Parameter	Notation
σ	Mobility rate of an MC
λ	Rate of queries from an MC for a data object
μ	Rate of updates of the server to a data object
δ	Aggregate rate of updates of all MCs but the one under consideration to a data object
η	Rate of updates of the MC under consideration to a data object
ω	Rate of reconnection when an MC switches from sleep mode back to active mode
α	Average distance (number of hops) between the gateway and an arbitrary MR
β	Average distance (number of hops) between the current serving MR of an MC after it wakes up and its serving MR before sleep
τ_c/τ_d	One-hop communication delay of transmitting a control/data packet between two MRs
γ_c/γ_d	Communication delay of transmitting a control/data packet between an MC and its serving MR
P_{hit}/P_{miss}	Cache hit/miss ratio

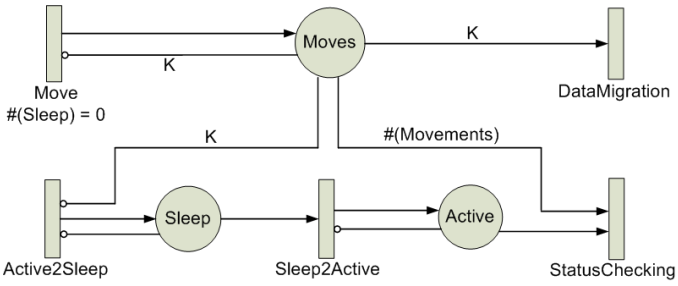

Fig. 4. The SPN model for DPM.

TABLE III
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
Active2Sleep	A timed transition modeling the state transition of the MC from active mode to sleep mode
Sleep	`Mark(Sleep)=1` means the MC is in sleep mode
Sleep2Active	A timed transition modeling the state transition of the MC from sleep 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 wakes up

- An MC typically switches alternatively between active mode and sleep 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 sleep mode. This is modeled by transition *Active2Sleep*, the transition rate of which is ω_s. After transition *Active2Sleep* is fired, a token is put into place *Sleep*, representing that the MC is switched to sleep mode. The MC wakes up and reconnects to the WMN after being in sleep mode for some time. This is modeled by transition *Sleep2Active*, the transition rate of which is ω_w.
- When the MC wakes up, it sends a query message to the data proxy where the data object currently 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 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*.

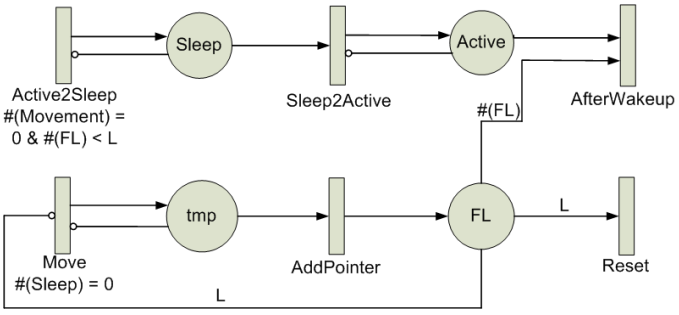

Fig. 5. The SPN model for CCM.

Fig. 5 depicts the SPN model for CCM. Table IV 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 σ. 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 serving MR becomes the MC's new forwarding chain head.
- Transitions *Active2Sleep* and *Sleep2Active* represent the same physical meanings as in the SPN model for DPM.

TABLE IV
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
Active2Sleep	A timed transition modeling the state transition of the MC from active mode to sleep mode
Sleep	`Mark(Sleep)=1` means the MC is in sleep mode
Sleep2Active	A timed transition modeling the state transition of the MC from sleep mode to active mode
Active	`Mark(Active)=1` means the MC is in active mode
AfterWakeup	A timed transition modeling the event of retrieving IRs from the forwarding chain head received during the MC's sleep and sending the gateway a location binding update message after the MC wakes up

- When the MC wakes up, it sends a query message to its current forwarding chain head to retrieve any IRs received by the forwarding chain head during its sleep. 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 *AfterWakeup* models the above events performed by the MC after it wakes up.

B. Performance Metrics

We use the *total communication cost incurred per time unit* and *access latency* as the metrics for performance evaluation and analysis. For DPM, the total communication cost incurred per time unit (C_{DPM}) consists of the query cost (C_q), the invalidation cost (C_i), the signaling cost of data migration (C_m), and the signaling cost of caching status checking upon reconnection (C_r). Using these cost terms, C_{DPM} is calculated as follows:

$$C_{DPM} = \lambda' \cdot C_q + (\mu + \delta + \eta') \cdot C_i + \sigma' \cdot C_m + \omega \cdot C_r \quad (1)$$

The various cost terms in the above equation can be calculated as follows (n_m denotes the number of tokens in place *Moves*; MC_0 represents the MC currently under consideration):

$$C_q = \begin{cases} n_m(\tau_c + \tau_d) + \gamma_c + \gamma_d & \textit{if cache hit} \\ (\alpha + n_m) \times (\tau_c + \tau_d) + \gamma_c + \gamma_d & \textit{if cache miss} \end{cases} \quad (2)$$

$$C_i = \begin{cases} 2\alpha\tau_c & \textit{other MCs} \\ 2(n_m + \alpha)\tau_c + 2\gamma_c & MC_0 \end{cases} \quad (3)$$

$$C_m = \begin{cases} K(\tau_c + \tau_d) + \gamma_c + \gamma_d & \textit{if cache hit} \\ 2K\tau_c + 2\gamma_c & \textit{if cache miss} \end{cases} \quad (4)$$

$$C_r = \begin{cases} \beta(\tau_c + \tau_d) + \gamma_c + \gamma_d & \textit{if cache hit} \\ 2\beta\tau_c + 2\gamma_c & \textit{if cache miss} \end{cases} \quad (5)$$

For CCM, the total communication cost incurred per time unit (C_{CCM}) consists of the query cost (C_q), the invalidation cost (C_i), the signaling cost of location management operations (C_l), and the signaling cost of cache status checking upon reconnection (C_r). Using these cost terms, C_{CCM} is calculated as follows:

$$C_{CCM} = \lambda' \cdot C_{query} + (\mu + \delta + \eta') \cdot C_i + \sigma' \cdot C_l + \omega \cdot C_r \quad (6)$$

The various cost terms in the above equation can be calculated as follows (f_l denotes the current forwarding chain length, and m_S denotes the number of tokens in place *Sleep*; MC_0 represents the MC currently under consideration):

$$C_q = \begin{cases} 0 & \textit{if cache hit} \\ (\alpha + l_f)\tau_d + \alpha\tau_c + \gamma_c + \gamma_d & \textit{if cache miss} \end{cases} \quad (7)$$

$$C_i = \begin{cases} 2\alpha\tau_c & \textit{server update} \\ 2\alpha\tau_c & \textit{other MCs} \\ (2\alpha + l_f)\tau_c + 2\gamma_c & MC_0 \end{cases} \quad (8)$$

$$C_l = \begin{cases} 2\tau_c + 2\gamma_c & \text{if } f_l < L \\ 2\alpha\tau_c + 2\gamma_c & \textit{otherwise} \end{cases} \quad (9)$$

$$C_r = 2(\alpha + \beta)\tau_c + 4\gamma_c \quad (10)$$

The calculation of some cost terms above depends on the cache hit ratio P_{hit}, which can be calculated as follows:

$$P_{hit} = \frac{\lambda'}{\lambda' + \mu + \delta + \eta'} \quad (11)$$

In the above equation, σ', λ', and η' denote the effective mobility rate, the effective data query rate, and the effective data update rate of the MC under consideration, respectively. These are effective rates because the MC cannot access or update any data object during its sleep. The calculation of these effective rates is shown below:

$$\begin{aligned} \sigma' &= P_{active}\sigma \\ \lambda' &= P_{active}\lambda \\ \eta' &= P_{active}\eta \end{aligned} \quad (12)$$

In the above equation, P_{active} denotes the probability that the MC can be found in active mode. Let ω_w and ω_s denote the rate of waking up and the rate of going to sleep, respectively, then P_{active} is calculated as follows:

$$P_{active} = \frac{\frac{1}{\omega_s}}{\frac{1}{\omega_s} + \frac{1}{\omega_w}} \quad (13)$$

An MC typically switches alternatively between active mode and sleep mode during its stay in a WMN. The rate of reconnection denoted by ω can be derived as follows:

$$\omega = \frac{\omega_w \times \omega_s}{\omega_w + \omega_s} \quad (14)$$

For both DPM and CCM, the access latency denoted as T is calculated as follows:

$$T = P_{hit} \times C_q^{hit} + (1 - P_{hit}) \times C_q^{miss} \quad (15)$$

where C_q^{hit} and C_q^{miss} denote the query cost under cache hit and cache miss, respectively, as given by Equation 2 for DPM, and Equation 7 for CCM.

V. 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 and access latency. We also carry out a comparative performance study to compare APPCCM with two non-adaptive that always cache a data object at the MC, or at the MC's current serving MR for cache consistency management in WMNs. It is worth noting that the total communication cost is on a per time unit, per MC basis, so even a performance difference of 5-10% will be significant over time. Below we use a parameter called *access to update ratio* (AUR) to model the data access/update pattern between an MC and the server. The AUR for a particular data object O_i is defined as: $AUR_i = \frac{\lambda_i}{\mu_i + \delta_i + \eta_i}$. We also define a parameter called *query to mobility ratio* (QMR) to represent the access/update characteristics with respect to O_i and the mobility characteristic of the MC. The QMR is defined as: $QMR_i = \frac{\lambda_i}{\sigma}$. Table V lists the parameters and their default values used in performance evaluation. These values are selected to demonstrate client-server applications with diverse data query and update characteristics, and mobile MCs characterized by diverse mobility and service patterns. The objective is to model a rich set of scenarios such that the adaptivity of APPCCM can be well presented and evaluated. We assume that future WMNs are based on advanced broadband wireless technologies, and the bandwidth among MRs is higher than that between MCs and MRs. The values of communication costs are chosen to reflect the assumption.

TABLE V
Parameters and their default values used.

Parameter	Value
QMR	{1, 2, 4, 8, 16, 32 ,64}
AUR	{0.125, 0.25, 0.5, 1.0, 2.0, 4.0, 8.0}
ω_w/ω_s	$\frac{1}{600}/\frac{1}{1200}$
α/β	20/5
τ_c/τ_d	0.001/0.002
γ_c/γ_d	0.002/0.004

A. Performance Evaluation of APPCCM

Fig. 6 plots the total communication cost incurred per time unit as a function of K/L in DPM and CCM, under two different values of AUR. As the figures illustrates, for either mechanism, there exists an optimal threshold K/L that minimizes the total communication cost incurred per time unit, given a set of parameters characterizing the mobility and access/update characteristics (with respect to the accessed data object) of the MC. As the figure shows, DPM performs consistently better than CCM, when $AUR = 0.5$, whereas

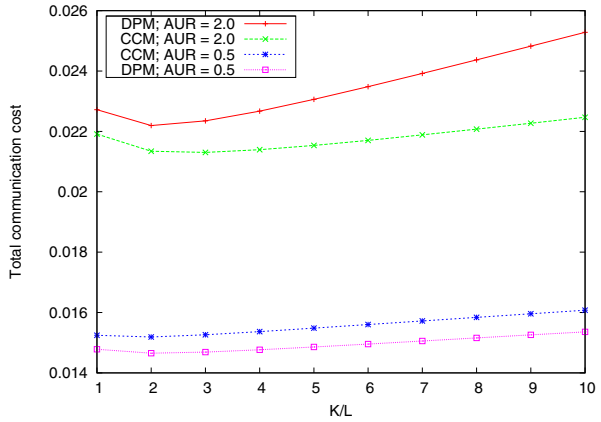

Fig. 6. Cost vs. K/L.

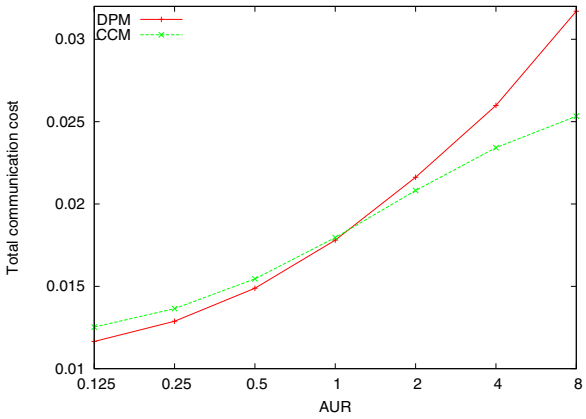

Fig. 7. Cost vs. AUR.

CCM is superior to DPM when $AUR = 2$. These results demonstrate the effect of the tradeoff between the query cost and invalidation cost on the performance of DPM and CCM.

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 with a price 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, with the price of increased invalidation costs. The effect of the tradeoff on the performance of DPM and CCM is clearly illustrated by Fig. 7, which compares the total communication cost incurred per time unit between DPM and CCM, as a function of AUR. As can be seen in the figure, initially when AUR is small, DPM performs better than CCM. As AUR increases, the performance gap between DPM and CCM decreases, and there exists a crossover point of AUR beyond which CCM becomes superior to DPM.

Fig. 8 plots the optimal threshold $K_{optimal}$ in DPM as a function of AUR. As shown in the figure, $K_{optimal}$ in DPM decreases with increasing AUR. This is because as AUR increases, the contribution of the query cost to the total communication cost becomes more significant, and therefore,

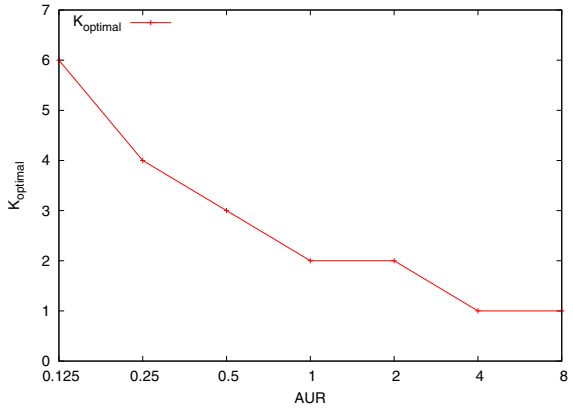

Fig. 8. $K_{optimal}$ vs. AUR in DPM.

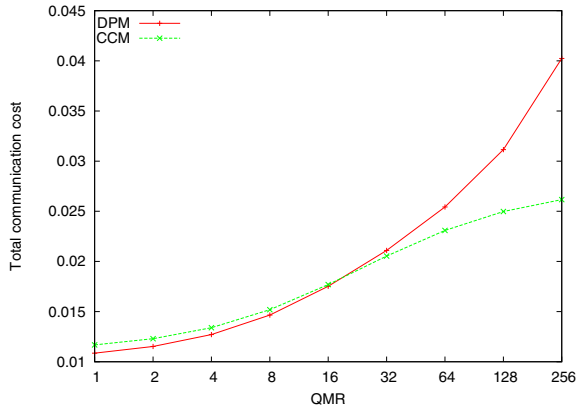

Fig. 9. Cost vs. QMR.

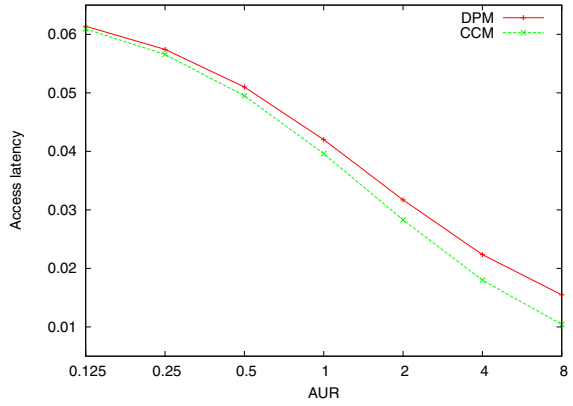

Fig. 10. Access latency vs. AUR.

a smaller $K_{optimal}$ is favored to reduce the query cost and consequently optimize the total communication cost.

Fig. 9 compares the total communication cost incurred per time unit between DPM and CCM, as a function of QMR. It can be observed that this figure shows the same trend as in Fig. 7. This is because with a fixed mobility rate and fixed update rates (μ, δ, and η), the access rate increases as QMR increases, and consequently AUR increases, resulting in the same trend as in Fig. 7.

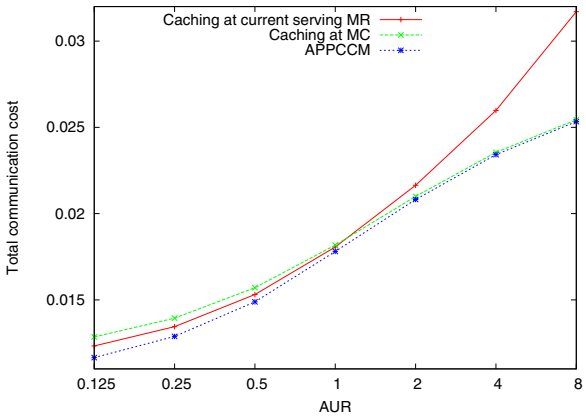

Fig. 11. Performance comparison: cost vs. AUR.

Fig. 10 compares the access latency between DPM and CCM, as a function of AUR. As the figure illustrates, CCM is consistently superior to DPM in terms of access latency under the same AUR. The reason is that DPM incurs additional cost for accessing a data proxy in a cache hit, compared to CCM that incurs little cost. Therefore, under the same cache hit ratio, CCM outperforms DPM in terms of access latency.

B. Performance Comparison

In this section, we compare APPCCM with two non-adaptive cache consistency management schemes that always cache a data object at the MC (caching-at-the-MC) or at the MC's current serving MR (caching-at-the-MR). Fig. 11 compares the total communication cost among APPCCM and the two non-adaptive schemes, as a function of AUR. As the figure shows, APPCCM outperforms both non-adaptive schemes. More specifically, APPCCM performs significantly better than the caching-at-the-MC scheme initially when AUR is small. As AUR increases, the performance gap decreases because APPCCM adaptively uses CCM to cache a data object when AUR is relatively large. Therefore, the advantage of APPCCM over the caching-at-the-MC scheme is most pronounced when AUR is relatively small. APPCCM is always superior to the caching-at-the-MR scheme. When AUR is small, APPCCM adaptively uses DPM to cache a data object. However, because APPCCM dynamically determines the optimal threshold $K_{optimal}$ under which the total communication cost is minimized, APPCCM performs significantly better than the caching-at-the-MR scheme that uses the constant threshold $K = 1$. It is worth emphasizing that because the total communication cost is a per time unit per MC metric, the accumulative effect of even a small performance gain is significant. These results demonstrate the advantage of APPCCM due to the exploitation of the tradeoff between the query cost and the invalidation cost.

VI. Conclusion

In this paper, we proposed APPCCM: an adaptive per-user per-object cache consistency management scheme for mobile Internet client-server applications in WMNs, to speedup data access as well as to mitigate the performance bottleneck at the gateway. APPCCM is adaptive, per-user and per-object, as one of two caching mechanisms provided by APPCCM, namely DPM and CCM, is adaptively selected to cache a data object based on the MC's mobility characteristic as well as its access/update characteristics with respect to the data object, and the WMN conditions. We demonstrated via model-based comparative 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 the exploitation of the tradeoff between the query cost and invalidation cost realized by adaptively selecting the optimal cache consistency management strategy out of two alternatives.

In the future, we plan to investigate community-based sharing of data objects in a data proxy among MCs currently being served by an MR. In addition, we plan to investigate how cooperative caching among MRs [2] can help improve the cache hit ratio, access latency, and overall network cost.

References

[1] Ian F. Akyildiz, Xudong Wang, Weilin Wang, "Wireless mesh networks: a survey," *Computer Networks*, vol. 47, no. 4, pp. 445–487, Mar. 2005.

[2] S. M. Das, H. Pucha, Y. C. Hu, "Mitigating the gateway bottleneck via transparent cooperative caching in wireless mesh networks," *Ad Hoc Networks*, vol. 5, no. 6, pp. 680–703, Aug. 2007.

[3] W. He, I. R. Chen, "A proxy-based integrated cache consistency and mobility management scheme for client-server applications in Mobile IP systems," *Journal of Parallel and Distributed Computing*, vol. 69, no. 6, pp. 559–572, Jun. 2009.

[4] D. Barbara, T. Imielinski, "Sleepers and workaholics: Caching strategies in mobile environments (Extended version)," *The VLDB Journal*, vol. 4, no. 4, pp. 567–602, Oct. 1995.

[5] J. Jing, A. Elmagarmid, A. S. Helal, "Bit-sequences: an adaptive cache invalidation method in mobile client/server environments," *Mobile Networks and Applications*, vol. 2, no. 2, pp. 115–127, Oct. 1997.

[6] Q. Hu, D. K. Lee, "Cache algorithms based on adaptive invalidation reports for mobile environments," *Cluster Computing*, vol. 1, no. 1, pp. 39–50, 1998.

[7] 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*, vol. 12, no. 7, pp. 686–700, Jul. 2001.

[8] G. Cao, "A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 5, pp. 1251–1265, Set./Oct. 2003.

[9] Y. B. Lin, W. R. Lai, J. J. Chen, "Effects of Cache Machanism on Wireless Data Access," *IEEE Transactions on Wireless Communications*, vol. 2, no. 6, pp. 1247–1258, Nov. 2003.

[10] Z. Wang, M. Kumar, S. K. Das, H. Shen, "Dynamic Cache Consistency Schemes for Wireless Cellular Networks," *IEEE Transactions on Wireless Communications*, vol. 5, no. 2, Feb. 2006.

[11] Y. Xiao, H. Chen, "Optimal Callback with Two-Level Adaptation for Wireless Data Access," *IEEE Transactions on Mobile Computing*, vol. 5, no. 8, Aug. 2006.

[12] A. Madhukar, T. Ozyer, R. Alhajj, "Dynamic cache invalidation scheme for wireless mobile environments," *Wireless Networks*, vol. 15, no. 6, pp. 727–740, Aug. 2009.

[13] K. L. Tan, J. Cai, B. C. Ooi, "An Evaluation of Cache Invalidation Strategies in Wireless Environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 8, pp. 789–807, Aug. 2001.

[14] 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*, accepted to appear, 2010.