Flexible Cache Consistency Maintenance over Wireless Ad Hoc Networks

Yu Huang, *Member*, *IEEE*, Jiannong Cao, *Senior Member*, *IEEE*, Beihong Jin, Xianping Tao, *Member*, *IEEE*, Jian Lu, *Member*, *IEEE*, and Yulin Feng

Abstract—One of the major applications of wireless ad hoc networks is to extend the Internet coverage and support pervasive and efficient data dissemination and sharing. To reduce data access cost and delay, caching has been widely used as an important technique. The efficiency of data access in caching systems largely depends on the cost for maintaining cache consistency, which can be high in wireless ad hoc networks due to network dynamism. Therefore, to make better trade-off between cache consistency and the cost incurred, it would be highly desirable to provide users the flexibility in specifying consistency requirements for their applications. In this paper, we propose a general consistency model called Probabilistic Delta Consistency (PDC), which integrates the flexibility granted by existing consistency models, covering them as special cases. We also propose the Flexible Combination of Push and Pull (FCPP) algorithm which satisfies user-specified consistency requirements under the PDC model. The analytical model of FCPP is used to derive the balance of minimizing the consistency maintenance cost and ensuring the specified consistency requirement. Extensive simulations are conducted to evaluate whether FCPP can satisfy arbitrarily specified consistency requirements, and whether FCPP works cost-effectively in dynamic wireless ad hoc networks. The evaluation results show that FCPP can adaptively tune itself to satisfy various user-specified consistency requirements. Moreover, it can save the traffic cost by up to 50 percent and reduce the query delay by up to 40 percent, compared with the widely used Pull with TTR algorithm.

Index Terms—Data dissemination and sharing, cache consistency, wireless ad hoc network.

1 Introduction

Wireless Ad hoc Networks have received a lot of attention due to their desirable features and potential applications in pervasive Internet access, outdoor assemblies, and disaster salvage [1], [13], [15], [19], [22]. Ad hoc networks can be quickly deployed and are easy to reconfigure, and are ideal for situations where installing an infrastructure is too expensive or infeasible. One major application of wireless ad hoc networks is to extend the Internet coverage and support pervasive and efficient data dissemination and sharing. In many scenarios, wireless terminals are spread over a large area in which access to external data is achieved through one or more access points (APs). For example, in an ad hoc network shown in Fig. 1, users close to an access point can directly access the Internet, and hence, can serve as gateway nodes, while other users access the Internet via the gateway nodes through multihop wireless connections. Ad hoc networks can be used to support pervasive Internet access in many scenarios, including university campuses, airports, and mobile stores [1], [13], [19].

The limited communication resources (e.g., bandwidth and battery power) and the opportunistic interaction among users make efficient data dissemination and sharing a challenging task in wireless ad hoc networks. Moreover, not all nodes have a direct link with the APs, and the AP may connect to an external network via a costly link (e.g., a satellite link). To reduce data access cost and delay, one widely used solution is to cache frequently accessed data at the data source node (gateway node) and a group of caching nodes [1], [2], [3], [22]. So, other users can access the cached data nearby with reduced traffic overhead and query delay.

In order to ensure valid data access, cache consistency [6], [13] must be maintained properly. Cache consistency refers to the consistency between the source data owned by the data source node and the cache copies held by the caching nodes. Maintaining cache consistency incurs overhead, which can be high in wireless ad hoc networks. Since different applications often have different requirements on the consistency level, it would be highly desirable to provide users the flexibility in specifying consistency requirements. Thus, users can efficiently trade cache consistency for reduced consistency maintenance cost.

Imagine that a group of people go hiking. They frequently need to access the Internet via their laptops or smart phones. A cost-effective way for Internet access is to form a wireless ad hoc network (as shown in Fig. 1), and cache popular data items on the gateway node and a collection of caching nodes. Users may have quite different requirements on how long the cached data has been stale. For example, the cached stock prices are required to be up-to-date within some short time frame. Users are willing to

- *Y. Huang, X. Tao, and J. Lu are with the Department of Computer Science and Technology, Nanjing University, 22# Hankou Road, Nanjing, Jiangsu Province 210093, China. E-mail: {yuhuang, txp, lj}@nju.edu.cn.*
- *J. Cao is with the Department of Computing, Hong Kong Polytechnic University, Kowloon, Hong Kong, China. E-mail: csjcao@comp.polyu.edu.hk.*
- *B. Jin and Y. Feng are with the Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, 4# Zhongguancun Nansijie, Beijing 100190, China. E-mail: {jbh, feng}@otcaix.iscas.ac.cn.*

Manuscript received 21 Feb. 2009; revised 6 Oct. 2009; accepted 13 Nov. 2009; published online 24 Dec. 2009.
Recommended for acceptance by M. Singhal.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2009-02-0080.
Digital Object Identifier no. 10.1109/TPDS.2009.168.

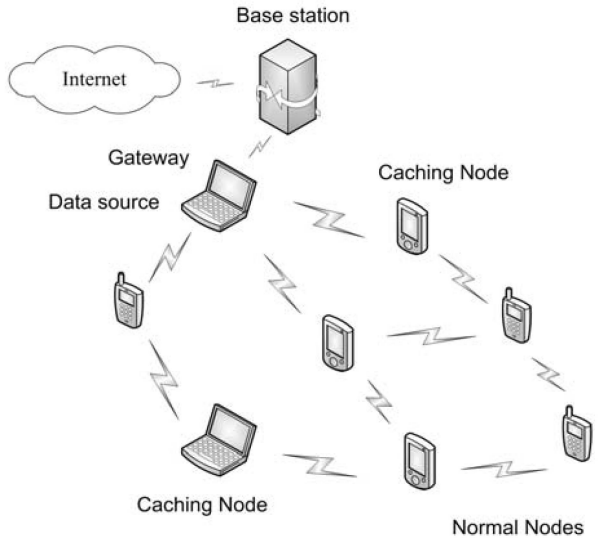

Fig. 1. A MANET for pervasive Internet access.

pay the cost, such as battery power and query delay, to access up-to-date stock prices. In contrast, users may tolerate, to certain degree, stale weather forecast information, since such information usually does not change dramatically in short time.

Meanwhile, users may not require that all accessed data objects must be consistent, especially when the cached data are frequently accessed. In the example above, the user may be eager to know the stock prices, and thus, check them frequently. In another scenario, taxies form a multihop vehicular ad hoc network to share the traffic information. The taxi driver may frequently check the traffic information so as to avoid the traffic jam. In such situations, as long as a certain portion of accessed data objects is consistent, the user can still accurately acquire the desired information.

Toward the objective of enabling users to make flexible trade-offs between cache consistency and consistency maintenance cost, two essential issues need to be addressed:

- Design of a consistency model, which enables the users to flexibly and precisely specify their consistency requirements.
- Design of a consistency maintenance algorithm, which satisfies user-specified consistency requirements at the minimum cost.

In the literature, Delta Consistency (DC) [6], [13] and Probabilistic Consistency (PC) [7], [20] provide a certain degree of flexibility for users to specify their consistency requirements, either in the maximum acceptable data deviation or in the guaranteed probability of valid data access. In this paper, we propose a general and flexible consistency model, called *Probabilistic Delta Consistency* (PDC), which covers all widely studied consistency models including DC and PC, as shown in Fig. 2 (formally defined in Section 3). In PDC, users can specify their consistency requirements in two orthogonal dimensions. The dimension along the x-axis specifies the value δ, which denotes the maximum acceptable deviation (in time, value, etc.) between the source data and the cache copies; the dimension along the y-axis specifies the probability p,

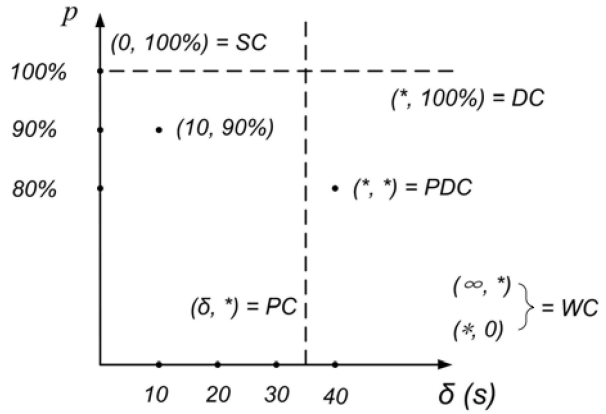

Fig. 2. The probabilistic delta consistency model.

which represents the minimum ratio of queries served by consistent cache copies.

To satisfy consistency requirements under the PDC model, we propose the *Flexible Combination of Push and Pull* (FCPP) algorithm. In FCPP, each cache copy is associated with a time-out value which is calculated based on the consistency requirements δ and p. Cache copies with valid time-out values can directly serve cache queries. Upon each update, the data source node sends an *invalidation* (INV) message to each cache copy possessing a valid time-out and requires an acknowledgement (INV_ACK message) of the invalidation. The data source node can update the source data if it has collected the INV_ACK messages for all the INV messages sent. Otherwise, it postpones updating the data until either the time-out values of all unresponding cache copies expire, or the maximum tolerable delay of data update is reached. FCPP is a generic and flexible scheme. By adjusting the time-out value associated with the cache copies, as well as the tolerable delay of updating the source data, FCPP covers many existing schemes, including *Lease* [9], [10], *Pull each read* [8], and *Push with ACK*, as its special cases.

The analytical model of FCPP is used to derive the balance of ensuring the specified consistency requirement and minimizing the consistency maintenance cost. In particular, the FCPP algorithm tunes to a dynamic *push-and-pull* algorithm when the cache query rate is high and the data update rate is low. Otherwise, FCPP tunes to an adaptive pull algorithm. Extensive simulations are conducted to evaluate whether FCPP can satisfy arbitrarily specified consistency requirements, and whether FCPP works cost-effectively in dynamic wireless ad hoc networks. The evaluation results show that FCPP effectively satisfies arbitrarily specified consistency requirements. The results also show that FCPP can save up to 50 percent of the traffic overhead and reduce the query delay by up to 40 percent, compared with the widely used *Pull with TTR* algorithm [1], [11], [12], [17]. A preliminary version of this paper is presented in [19].

The rest of this paper is organized as follows: Section 2 provides an overview of the existing work. Section 3 presents definition of the PDC model. In Section 4, we present design of the FCPP algorithm. An analytical model is derived in Section 5. Section 6 presents the experimental evaluation. Finally, Section 7 concludes the paper with a summary and the future work.

2 Related Work

There are various levels of cache consistency. Two extremes of the *consistency spectrum* are Strong Consistency (SC) and Weak Consistency (WC). In SC, the accessed cache copies are always up-to-date, whereas in WC, cache consistency is maintained in a best-effort manner [6], [13]. The weak consistency model is principally the same with the Eventual Consistency (EC) model [23]. To satisfy consistency requirements between SC and WC, DC [6], [13] and PC [7], [20] have been proposed. In DC, users can specify the maximum acceptable deviation between the source data and the cached data, while in PC, users can specify the probability of valid data access. DC and PC enable users to specify their consistency requirements in two orthogonal dimensions. Our PDC model integrates these two models into a uniform model.

To meet different consistency requirements, many consistency maintenance algorithms have been proposed. For example, Pull each read [8], Invalidation [4], Lease [9], [10], and UIR-based Cache Invalidation [16] were proposed to provide SC. The Predictive Caching Consistency algorithm [14] provides WC. However, these schemes cannot enable users to flexibly specify their consistency requirements. A widely used technique to support DC is to associate a Time to Refresh (TTR) value with each cache copy [1], [11], [12], [17]. In order to work efficiently in dynamic environments, the TTR value is dynamically adjusted [11], [12]. In [17], the authors proposed the Push-and-Pull and Push-or-Pull algorithms. In [13], a relay peer-based algorithm was proposed to provide different consistency levels, including SC, DC, and WC. However, none of the aforementioned algorithms support probabilistic consistency requirements. In [7], an adaptive pull algorithm was proposed to support PC, but it is designed only for numerical data values. So far, to the best of our knowledge, there is no single cache consistency maintenance algorithm which can effectively satisfy different consistency requirements, including SC, WC, DC, and PC, on various types of data objects. This work aims to propose such an algorithm.

3 Probabilistic Delta Consistency

In this section, we first present definition of the PDC model. Then, we show, by examples, how the PDC model enables users to flexibly specify their consistency requirements. Finally, we discuss the generality of PDC.

3.1 Definition of PDC

Let S^t denote version number of the source data and C_j^t denote that of the cache copy on node j at time t. Initially, S^t is set to *zero* and is increased upon each update. C_j^t is set to version number of the source data at the caching node's synchronization time [6]. The consistency requirement PDC(δ, p) is defined as, for any node j and any time t, the probability that the cache copy is stale within δ seconds is no less than p [1]:

$$\forall t, \forall j, \Pr\left\{\exists \tau : 0 \leq \tau \leq \delta :: S^{t-\tau} = C_j^t\right\} \geq p.$$

1. Delta Consistency can be defined in different domains, e.g., the time domain [6], [13] or the value domain [7]. Hence, the Probabilistic Delta Consistency model can also be defined in different domains. In this paper, we focus on the time domain.

3.2 Applying PDC

Users can flexibly specify their consistency requirements under the PDC model in different scenarios. For example, in the scenario discussed in Section 1, the user accessing stock prices can assure that the cached prices will not be too stale by specifying a small δ (e.g., 1 minute). Since he frequently checks the prices, he might be able to tolerate that a small portion of accesses not satisfying the requirement on δ. He can specify a p value slightly less than 100 percent (e.g., 95 percent).

Meanwhile, the user accessing the weather forecast information can specify a large δ such as 2 hours, since the weather forecast information is relatively stable. The user may not check the weather information frequently. So, he may set a high p value such as 90 percent.

In the other scenario, since the traffic information cannot change dramatically in a short time and the taxi driver may frequently access the traffic information, the driver may specify less stringent requirement on both the deviation δ (e.g., 10 minutes) and the ratio p (e.g., 70 percent).

3.3 Generality of PDC

The PDC model covers existing consistency models as its special cases, as shown in Fig. 2. PDC(0, 100 percent) is equivalent to the Strong Consistency model. $PDC(\infty, *)$ and PDC(*, 0) yield the Weak Consistency model ("*" represents any possible value). PDC(*, 100 percent) yields the Delta Consistency model. For specified δ, PDC$(\delta, *)$ yields the Probabilistic Consistency model.

4 The Flexible Combination of Push and Pull Algorithm

In this section, we first discuss the design aspects explored in the FCPP algorithm and present the detailed design. Then, we discuss the generality and flexibility of FCPP. We consider a commonly used system model [13], [19], where each data object is associated with a single *data source node*. Only the data source node can update the *source data*. Each data object is cached by a collection of *caching nodes*. The data copies held by the caching nodes are called *cache copies*. There are two basic mechanisms for cache consistency maintenance: *push* and *pull*. Using push, the data source node proactively informs the caching nodes of cache information. Using pull, a caching node fetches cache information from the data source node. We also assume that the data source node and the caching nodes have synchronized clocks.

4.1 Design Aspects

In designing FCPP, we focus on three design aspects, as discussed in detail below:

- *Consistency level*. To enable users to flexibly trade cache consistency for reduced cost, we need to provide them multiple consistency levels with fine granularity. We have addressed this issue in design of the PDC model. PDC enables the users to continuously tune their consistency requirements in two orthogonal dimensions.
- *Update delay*. Existing schemes mainly focus on how cache consistency should be maintained after the data source node has updated the source data. In

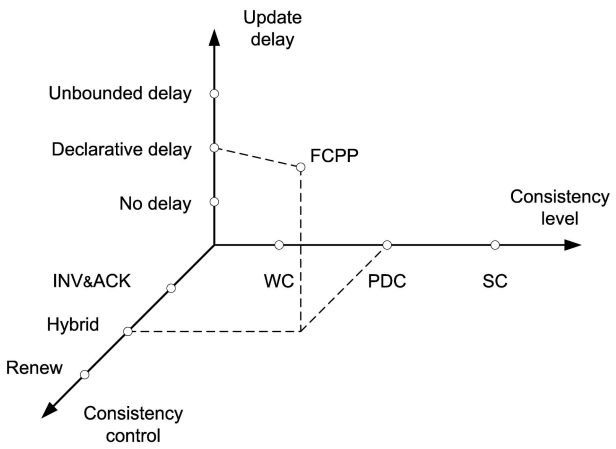

Fig. 3. The 3D design framework.

such schemes, the data source node can directly update the data without considering consistency maintenance. However, in many cases, the data source node can wait for certain time before updating the source data, as in the Two Phase Commit (2PC) protocol [24] and the *Lease* protocol [9], [10]. The update delay can be utilized to further decrease the consistency maintenance cost, as in *Lease*. However, *Lease* does not provide any bound on how long the data source node needs to wait before updating the source data. Between the two extremes of *no update delay* and *unbounded update delay*, FCPP supports declarative update delay (denoted by a system parameter D).

- *Consistency control*. A simple but widely used approach to consistency control is to associate time-out values with cache copies. When the time-out values expire, the caching nodes *renew* the time-out values from the data source node. Upon a data update on the data source node, it first needs to push *invalidations* to the caching nodes. Since the invalidations may be lost, especially in dynamic wireless ad hoc networks, the caching nodes are required to send back the *acknowledgments*. Both schemes (Renew and INV&ACK) are adaptively combined in FCPP.

We illustrate the design aspects and issues in each aspect with a 3D framework, as shown in Fig. 3. Based on our exploration of the issues in each design aspect, we present the detailed implementation of FCPP in Algorithm 1 and Algorithm 2.

Algorithm 1. FCPP on a caching node
Upon receiving a query
(1) IF($l > 0$) Serve the query with the cache copy;
(2) ELSE // *l has decreased to zero*
(2.1) Send a RENEW message to renew the timeout from the data source node and update the cache copy if necessary;
(2.2) After l is renewed, serve the query with the updated cache copy;

Algorithm 2. FCPP on the data source node
When the source data is ready to be updated
(1) Send an INV message to each caching node with positive l;
(2) IF(receive an INV_ACK message for each INV message OR has waited for D seconds)
(2.1) Update the source data;
Upon receiving a RENEW message from a caching node
(3) IF(the source data has been updated)
(3.1) Send data update to the caching node;
(4) IF (no update is pending)
(4.1) Grant timeout with duration l and the data update to the caching node;

4.2 FCPP as a Generic and Flexible Scheme

FCPP is a generic scheme, covering many existing schemes as its special cases:

- When the data source node can always delay the update until it receives all the ACKs, or the time-out values on all unresponding caching nodes expire, FCPP provides Strong Consistency and covers the *Lease* protocol [9], [10] as its special case.
- When the time-out value l is set to *zero*, FCPP transforms to the *Pull each read* [8] scheme.
- When the data source node lets l be sufficiently large, FCPP transforms to the *Push with ACK* scheme.

The key issue in tuning FCPP is to quantify the trade-off between consistency requirement PDC(δ, p) and the consistency maintenance cost. We study this issue in Section 5.

5 Analysis of the Optimized Combination of Push and Pull

In this section, we first derive an analytical model to study the relationship between the time-out value l and the consistency requirement PDC(δ, p). We also study the relationship between l and the consistency maintenance cost. Then, the optimized trade-off between cache consistency and the maintenance cost is derived:

- In case of high cache query rate and low data update rate, we calculate the optimized time-out duration l, which tunes FCPP to a *push-and-pull* algorithm (Section 5.2).
- In case of low cache query rate and high data update rate, FCPP is tuned to an adaptive pull algorithm (Section 5.3).

In the analytical model, we assume that data update requests from the application and cache queries follow the *Poisson Process*. The hop count of consistency maintenance message transmission is used to measure the cost [21]. Notations used in the analytical model are listed in Table 1.

5.1 Satisfying Consistency Requirement $\mathrm{PDC}(\delta, p)$

We first argue that when $l \leq D + \delta$, no stale cache hits occur in the sense of Delta Consistency. To see this, suppose that a data update occurs at time t_u, as shown in Fig. 4. According to the design of FCPP, when a data update is pending, no time-out value is granted to the caching nodes. As for time-out values granted before the update, they may be still valid on some caching nodes. The data source node first sends an

TABLE 1
Notations Used in the Analytical Model

(δ, p)	user-specified consistency requirement
D	maximum time the data source node can wait before updating the source data
l	timeout duration
N	number of caching nodes
N_k	the k^{th} caching node, k = 1, 2, …, N
S	expected number of stale cache hits on one caching node
ES	expected number of stale cache hits on all caching nodes
$I(k)$	indicator random variable $I(k)$ = 1, if the timeout on N_k is valid and the INV_ACK message from N_k is missed; Otherwise, $I(k)$ = 0
p_k	probability that the INV_ACK message from N_k is missed
$\overline{h}$	average path length between the data source node and the caching nodes
w	average data update rate
r	average cache query rate
t_u	time instant of a source data update
(t_1, t_2)	period between time instants t_1 and t_2
C	total consistency maintenance cost per unit time
$P_{update}(k, t_1, t_2)$	the probability that there are k source data updates in period (t_1, t_2)

INV message to each caching node possessing a valid time-out. Then, it waits for at most D seconds. In the worst case, every INV message is lost and every caching node has the maximum time-out duration $l = D + \delta$, inducing maximum stale cache hits. Observe that:

- During period $(t_u,\ t_u + D)$, there are no stale cache hits, since the data source node has not updated the source data yet.
- During period $(t_u + D, t_u + D + \delta)$, there are no stale cache hits either, since the deviation between the source data and the cache copy is bounded by δ.

Thus, we show that there are no stale cache hits when $l \leq D + \delta$.

Then, we calculate the expected number of stale cache hits when $l > D + \delta$. Upon a data update at t_u, the time-out value remained on the caching node (denoted by x) lies in range $[0, l]$:

- If x is in range $[0, D + \delta]$, there will be no stale hits, as discussed above.
- If x is in range $[D + \delta, l]$ while this caching node misses the INV message and still serves cache queries, the expected number of stale hits is $(x - D - \delta)r$.

We average the number of stale hits in both cases over range $[0, l]$ and obtain the expected number of stale hits on one caching node in period $(t_u, t_u + l)$:

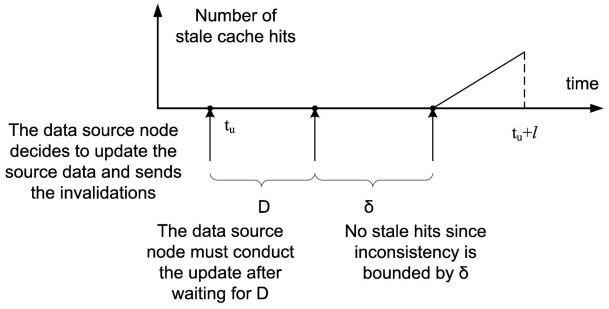

Fig. 4. Number of stale cache hits.

$$S = \frac{1}{l}\int_{\delta+D}^{l}(x - \delta - D)r dx = \frac{r}{2l}(l - \delta - D)^2.$$

When the data source node misses the INV_ACK message from a caching node, two cases may occur: 1) the caching node misses the INV message, or 2) the caching node has successfully received the INV message, but the corresponding INV_ACK message is lost. In the first case, the data source node cannot receive the corresponding INV_ACK message. The expected number of stale cache hits introduced by this caching node is S obtained above. In the second case, the data source node will falsely decide that the caching node introduces stale cache hits. Our estimation of the number of stale cache hits is conservative in this case. Concerning both cases, we have that the number of stale hits always satisfies the inequality $S \leq \frac{r}{2l}(l - \delta - D)^2$ no matter what the value of l is.

Based on the bound on the number of stale hits on one caching node, the total number of stale hits on all caching nodes is bounded by:

$$ES \leq \sum_{1 \leq k \leq N} p_k \cdot I(k) \cdot S \leq \frac{r}{2l}(l - \delta - D)^2 \sum_{1 \leq k \leq N} p_k \cdot I(k). \quad (1)$$

Here, p_k denotes the probability that the INV_ACK message from N_k is missed. For each caching node $N_k (1 \leq k \leq N), \mathrm{I}(k) = 1$, if N_k has a time-out l greater than $D + \delta$ and the INV_ACK message from N_k is missed. Otherwise, $I(k) = 0$. When the data source node is ready to update the data, it can determine the value of $I(k)$ for each caching node based on the values of l and δ.

With estimation of the expected number of stale hits ES, we further relate this number with the consistency requirement $\mathrm{PDC}(\delta, p)$. Upon each source data update, the data source node sets the time-out value to l. The average interval between the current and the forthcoming source data update is $1/w$. For N caching nodes in total, the expected number of cache queries in this period is $r \cdot N/w$. With the expected number of stale cache hits ES, the probability of Delta Consistency is $1 - (w \cdot ES)/(r \cdot N)$. Since this probability should be no less than the user-specified p, we have that $p \leq 1 - (w \cdot ES)/(r \cdot N)$, which is equivalent to: $ES \leq (1 - p)N \cdot r/w$. Based on the upper

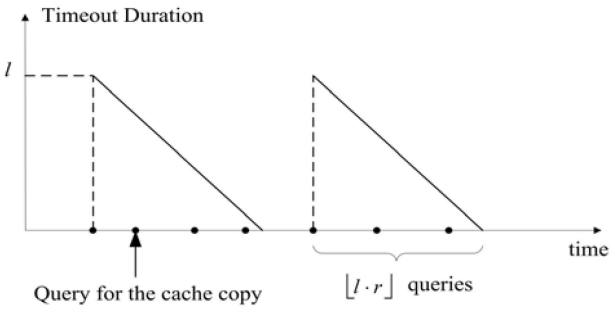

Fig. 5. Number of queries within the time-out duration.

bound of *ES* in Inequality (1), we can satisfy the consistency requirement $\mathrm{PDC}(\delta, p)$ if we ensure that:

$$\frac{r}{2l}(l-\delta-D)^2 \sum_{1\le k\le N} p_k \cdot I(k) \le \frac{(1-p)N \cdot r}{w}.$$

To solve this inequality and obtain the upper bound of l, note that $\mathrm{d}(l-D-\delta)^2/\mathrm{d}l > 0$ when $l > D+\delta$. This shows that the maximum value of l is the root of equation:

$$\sum_{1\le k\le N} p_k I(k) \frac{r}{2l}(l-\delta-D)^2 = \frac{(1-p)N \cdot r}{w}.$$

We solve this quadratic equation and obtain the upper bound of l:

$$l \le (D+\delta) + \frac{(1-p)N}{w\sum_{1\le k\le N} p_k I(k)} + \sqrt{\frac{(1-p)^2N^2}{\left(w\sum_{1\le k\le N} p_k I(k)\right)^2} + \frac{2(D+\delta)(1-p)N}{w\sum_{1\le k\le N} p_k I(k)}}. \quad (2)$$

5.2 Consistency Maintenance Cost

There are mainly two types of consistency maintenance cost induced by FCPP:

- *Cost for time-out renewal.* As shown in Fig. 5, when a query comes after the time-out value expires, the caching node first renews the time-out value to l, and then, serves the next $l \cdot r$ (on average) queries. Thus, for every $l \cdot r + 1$ queries, there will be one time-out renewal. So, we average the cost over the queries and obtain the expected renewal cost per unit time: $2 \cdot \overline{h} \cdot N \cdot r/(l \cdot r + 1)$.
- *Cost for the INV & ACK process.* Concerning the data updates, for every $l \cdot r + 1$ queries on a cache copy, $l \cdot r$ of them occur when the time-out value is valid. The probability of having a valid time-out value is $l \cdot r/(l \cdot r + 1)$. As long as the time-out value is valid, the data source node needs the INV & ACK process upon a data update. Thus, the expected INV & ACK cost per unit time is: $2 \cdot \overline{h} \cdot N \cdot w \cdot l \cdot r/(l \cdot r + 1)$.

According to the discussions above, we obtain the total consistency maintenance cost per unit time:

$$C = 2N \cdot \overline{h}\frac{r + w \cdot l \cdot r}{l \cdot r + 1} = 2r \cdot N \cdot \overline{h}\frac{1 + w \cdot l}{l \cdot r + 1}.$$

In order to investigate the impact of time-out value l on the cost C, we take the derivation:

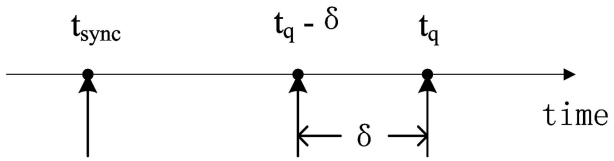

Fig. 6. Deciding to pull or not on a caching node.

$$\frac{dC}{dl} = 2r \cdot N \cdot \overline{h} \cdot (w - r)/(l \cdot r + 1)^2.$$

When $w < r$ (higher cache query rate and lower data update rate), we have $dC/dl < 0$, which means that the time-out duration should be increased as much as possible to minimize the consistency maintenance cost.

Combining the result of both Sections 5.1 and 5.2, we obtain that the optimal time-out value (satisfying the consistency requirement while minimizing the maintenance cost) when $w < r$ should be:

$$l = (D+\delta) + \frac{(1-p)N}{w\sum_{1\le k\le N} p_k I(k)} + \sqrt{\frac{(1-p)^2N^2}{\left(w\sum_{1\le k\le N} p_k I(k)\right)^2} + \frac{2(D+\delta)(1-p)N}{w\sum_{1\le k\le N} p_k I(k)}}. \quad (3)$$

5.3 Dealing with High Data Update Rate and Low Cache Query Rate

When the data update rate is no less than the cache query rate ($w \ge r$), the time-out duration l should be set to *zero*, since $\mathrm{d}C/\mathrm{d}l > 0$, as discussed above. Hence, the data source node can directly update the source data without any interaction with the caching nodes. On the caching node side, if the caching nodes always pull the data source node upon each query, Strong Consistency is strictly guaranteed. In this case, the user-specified consistency requirement $\mathrm{PDC}(\delta, p)$ is oversatisfied and the round-trip consistency maintenance cost imposed by the *Pull Each Read* algorithm should be reduced. We study in this section how the caching nodes adaptively decide whether to pull the data source node, in order to satisfy user-specified consistency requirement $\mathrm{PDC}(\delta, p)$, while minimizing the consistency maintenance cost.

Suppose that a cache query comes at time t_q, and the latest synchronization between the cache copy and the source data occurs at t_{sync}, as shown in Fig. 6. We find that: 1) if $t_q - \delta \le t_{sync}$, the caching node can directly serve cache queries while strictly guaranteeing Delta Consistency. It is because in this case, the deviation between the source data and the cache copy is bounded by δ. 2) If $t_q - \delta > t_{sync}$, Delta Consistency cannot be guaranteed if the caching node directly serves cache queries.

Observe that Delta Consistency is guaranteed if and only if there are no source data updates in period (t_{sync}, $t_q - \delta$). We can ensure the user-specified consistency requirement $\mathrm{PDC}(\delta, p)$ if it is guaranteed that the probability $\mathrm{P_{update}}(0, t_{sync}, t_q - \delta)$ is more than the user-specified probability p. Since the source data update follows the *Poisson Process*, we have that:

$$\mathrm{P_{update}}(\mathrm{k}, \mathrm{t_1}, \mathrm{t_2}) = [\lambda_\mathrm{u}(\mathrm{t_2} - \mathrm{t_1})]^\mathrm{k} \mathrm{e}^{-\lambda(\mathrm{t_2}-\mathrm{t_1})}/\mathrm{k}!.$$

To satisfy the consistency requirement $\mathrm{PDC}(\delta, p)$, the following inequality should be satisfied:

$$\mathrm{P}_{\mathrm{update}}(0, \mathrm{t}_{\mathrm{sync}}, \mathrm{t}_{\mathrm{q}} - \delta) = \mathrm{e}^{-\lambda_{\mathrm{u}}(\mathrm{t}_{\mathrm{q}} - \delta - \mathrm{t}_{\mathrm{sync}})} \geq \mathrm{p}. \quad (4)$$

Based on inequality (4), the caching node can make online decisions about whether to pull. According to the analysis above, the caching node will pull only when the consistency requirement would be violated without pull. Thus, the adaptive pull strategy based on inequality (4) also yields optimized consistency maintenance cost.

5.4 Implementation of FCPP

Concerning the implementation of FCPP, parameter D is initialized upon deployment of the caching system, and users specify their consistency requirement PDC(δ, p). Initially, l is set to $D + \delta$ and Delta Consistency is strictly guaranteed. After the FCPP algorithm is launched for a certain period of time, the data source node obtains the source data update rate w and the cache query rate r on each caching node. Then, the data source node can decide the relationship between w and r. Upon each time-out RENEW request, the data source node compares w with r to tune the FCPP algorithm:

- If $w < r$, the data source node calculates the optimized time-out value l with (3), which is granted to each caching node. The caching nodes and the data source node employ Algorithms 1 and 2, respectively, to satisfy the consistency requirements.
- If $w \geq r$, the data source node directly updates the source data. The data source node sets l to a special value (e.g., -1), which makes the caching nodes adaptively decide whether to pull based on inequality (4).

6 Experimental Evaluation

In this section, extensive simulations are conducted to evaluate FCPP. We first present the experimental methodology and configurations. Then, we discuss the evaluation results.

6.1 Experiment Methodology

The primary contribution of this work is to enable the optimized trade-off between cache consistency and consistency maintenance cost. Thus, our evaluation consists of two parts, focusing on both sides of the trade-off. We first study whether FCPP can satisfy arbitrarily specified consistency requirements under the PDC model. Then, we study the cost-effectiveness of FCPP by extensive performance comparison. Note that the performance comparison should be conducted under the same consistency requirement. Since no existing algorithms can satisfy arbitrarily requirement $\mathrm{PDC}(\delta, p)$, we set the consistency requirement to Delta Consistency, i.e., the special case (δ, 100 percent) under the PDC model.

FCPP is compared with the *Pull with TTR* algorithm, which provides Delta Consistency [1], [11], [12], [17]. Pull with TTR is the most widely used scheme for Delta Consistency in existing caching systems. The performance comparison is conducted in different network settings with different data update rate, network size, and node mobility. These parameters are varied because they have the most significant impact on the performance of FCPP. The following performance metrics are used in the evaluation:

- (δ, p): consistency requirement under the PDC model.
- *Traffic overhead*: average hop count of consistency maintenance message propagation.
- *Query delay*: average delay (measured in milliseconds) imposed due to consistency maintenance.

6.2 Experiment Configurations

In the experiments, 80 mobile hosts are scattered to a rectangular area of size $200 \times 200\ \mathrm{m}^2$. Users move around in the territory following the *random waypoint* model and the *Gauss-Markov* model [18]. Transmission range of the mobile hosts is set to 15 m. Up to 10 percent of mobile hosts will gradually crash at randomly chosen time instants. Each hop of message transmission has loss rate 5 percent. The routing protocol adopts the *least hops* routing metric. We focus on the cost for consistency maintenance. The cost for setting up and maintaining the routing tables is not counted. The tolerable update delay of the data source node is 2 s. Data updates and cache queries are generated with the Poisson process. The average interval of cache queries is 5 s. The consistency requirement δ and p are varied in range 5-20 s, and 60-90 percent, respectively. The update interval is varied between 20 and 80 s. The number of caching nodes is varied from 10 to 70, and the node speed is varied from 0.5 to 2.0 m/s. The default values of these parameters are set as follows:

- (δ, p): (5 s, 90 percent);
- Average update interval: 20 s;
- Number of caching nodes: 10;
- Node speed: 0.5 m/s.

Detailed experimental configurations are listed in Table 2.

6.3 Satisfying User-Specified Consistency Requirements

In this experiment, we tune the consistency requirement in two dimensions. First, p is decreased from 90 to 60 percent. We find that when the value of p is high, the actual probability is slightly more than the user-specified probability, as shown in Fig. 7. The difference between actual p and specified p is less than 3 percent, when the specified p is 90 percent and 80 percent. When specified p is decreased to 70 percent and 60 percent, the difference between both probabilities goes up to 15 percent. This shows that the estimation of FCPP is less accurate when the specified probability p is low. It is mainly because when the specified p is decreased, the cache copies are associated with larger time-out values. In this case, there are more chances for the data source node to conservatively calculate the expected number of stale cache hits.

As for the consistency maintenance cost, we find that the traffic overhead and the query delay decrease approximately in a linear manner when p is decreased, as shown in Figs. 8 and 9. When the value of specified p is low, the traffic overhead and the query delay decrease a bit more slowly. The traffic overhead and query delay decrease by 26 and 41 percent, respectively, when the consistency requirement p is relaxed from 90 to 60 percent. The

TABLE 2
Experimental Configurations

Network area	$200 \times 200\ m^2$
Size of network	80
Number of caching nodes	10 ~ 70
Transmission range	15 *m*
Mobility model	*Random waypoint, Gauss-Markov*
Average speed	0.5 *m/s* ~ 2.0 *m/s*
Maximum portion of crashed nodes	10%
Routing metric	*Least hops*
Probability of message loss per hop	5%
Consistency requirement δ	5 *s* ~ 20 *s*
Consistency requirement p	60% ~ 90%
Maximum delay before data update	2 *s*
Pattern of date updates and queries	*Poisson process*
Average update interval	20 *s* ~ 80 *s*
Average query interval	5 *s*

decrease in consistency maintenance cost when the consistency requirement is relaxed guarantees that the users can trade cache consistency for reduced cost.

Then, we increase the value of δ from 5 to 20 s. We find that the actual probability is more close to specified probability when δ is less, as shown in Fig. 10. FCPP becomes more accurate since the caching nodes contact the data source node more frequently with less δ values.

When the consistency requirement δ is relaxed from 5 to 20 s, the traffic overhead and the query delay also decrease approximately in a linear manner. The traffic overhead and the query delay decrease by 38 and 51 percent, respectively, as shown in Figs. 11 and 12. We show that users can also relax their requirements on δ to reduce the consistency maintenance cost.

In summary, we find, in this experiment, that FCPP can satisfy arbitrarily specified consistency requirements. In particular, FCPP achieves more accurate estimation when the consistency requirement is more stringent. We also find that the consistency maintenance cost decreases linearly as users' consistency requirements decrease, which guarantees that users can trade cache consistency for reduced cost. In the next experiment, we further study the cost-effectiveness of FCPP in different environment settings.

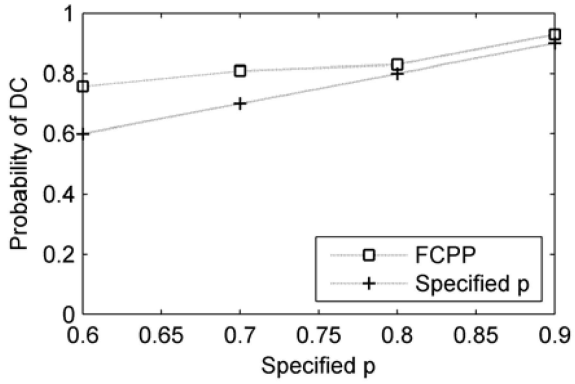

Fig. 7. Actual p versus specified p.

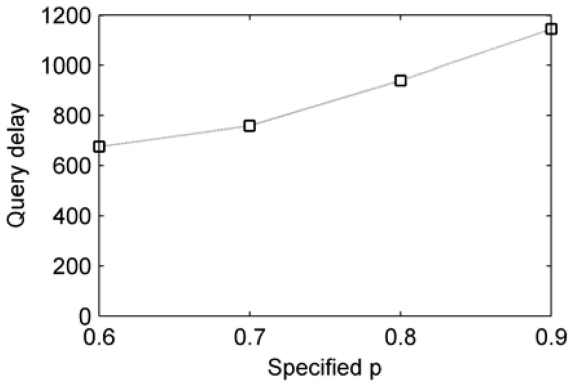

Fig. 9. Query delay versus p.

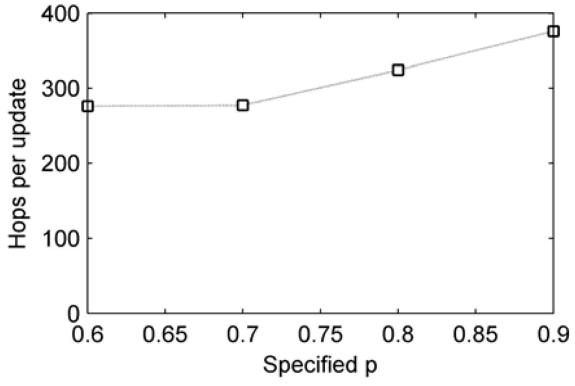

Fig. 8. Traffic overhead per update versus p.

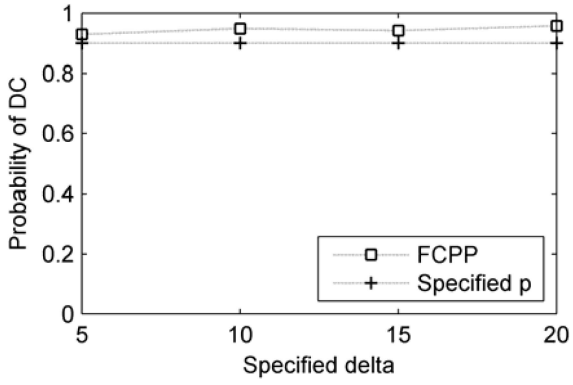

Fig. 10. Probability of DC versus δ.

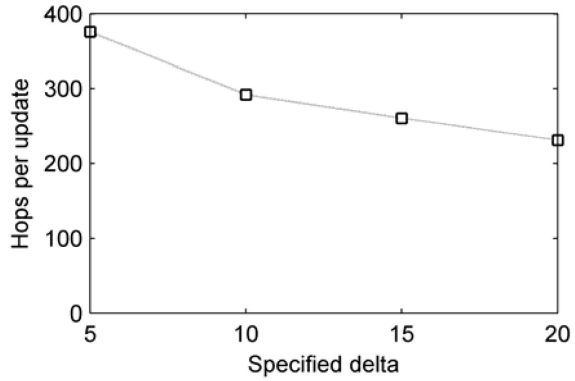

Fig. 11. Traffic overhead per update versus δ.

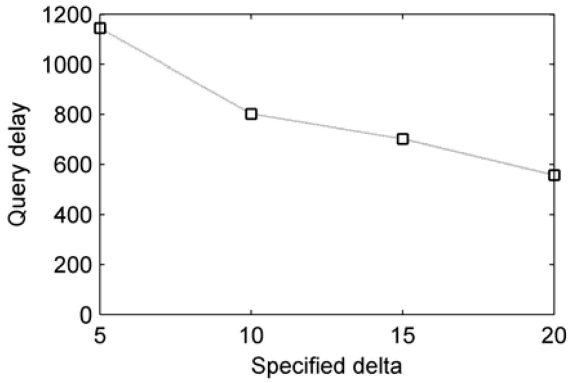

Fig. 12. Query delay versus δ.

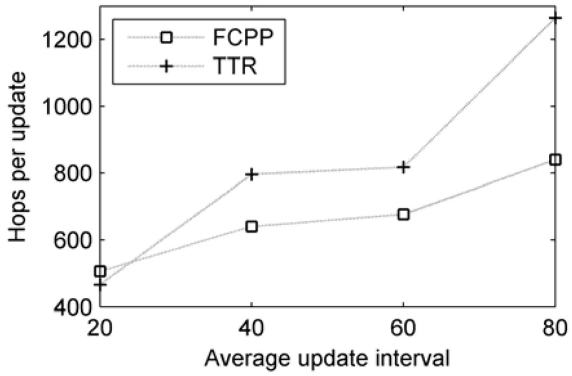

Fig. 13. Traffic overhead per update versus update interval.

6.4 Cost-Effectiveness and Adaptability in Dynamic Environments

As discussed in Section 6.1, we compare FCPP with the *Pull with TTR* algorithm in the performance comparison. Both schemes are required to provide Delta Consistency ($\delta = 5$ s). The performance comparison is conducted in dynamic environments with different data update rate, number of caching nodes, as well as node mobility level and mobility model.

6.4.1 Dealing with Different Data Update Rate

In this experiment, we increase the update interval from 20 to 80 s. We find that FCPP induces slightly more traffic overhead than Pull with TTR, when faced with frequent updates ($\text{update interval} = 20$, Fig. 13). This is mainly due to the round-trip traffic overhead of the INV & ACK process.

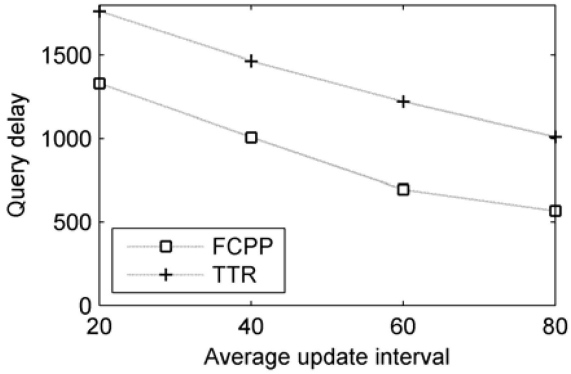

Fig. 14. Query delay versus update interval.

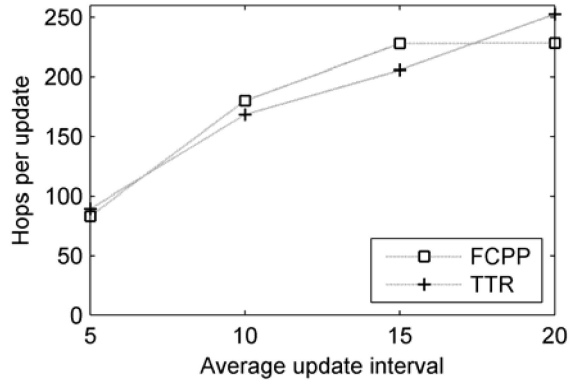

Fig. 15. Traffic overhead per update versus update interval.

Though Pull with TTR is a simple pull scheme. It does not induce any overhead on the data source node upon updates, thus being able to effectively deal with frequent data updates.

As the data update interval increases, FCPP becomes much more effective. It reduces the traffic overhead by 50 percent when faced with infrequent updates ($\text{update interval} = 80$, Fig. 13). This is mainly because the cost of the (infrequent) INV & ACK process in FCPP is well compensated when the caching nodes directly serve the (frequent) cache queries. Note that when the update rate gets lower while the query rate holds, more queries are associated with one update, and the traffic overhead per update increases.

As for the query delay, FCPP is better than Pull with TTR by around 40 percent, and the gap between the delay imposed by both schemes is relative stable when the update interval changes, as shown in Fig. 14. The delay is mainly decided by how many cache queries can be directly served by the caching nodes. FCPP is better since it sets optimized time-out values based on its analytical model, while Pull with TTR sets static time-out values.

We also study the cost-effectiveness of FCPP when the update rate is higher than the query rate. We first set the update interval to 20 s, and then, decrease it to 5 s. The cost induced by both schemes is similar, as shown in Figs. 15 and 16. This is mainly because FCPP transforms to a pull-based scheme when faced with more frequent updates and less frequent cache queries, as discussed in Section 5.3. The difference between the consistency maintenance cost is mainly due to the dynamism of wireless ad hoc networks.

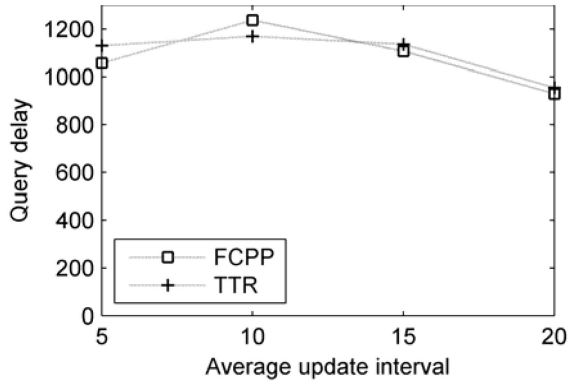

Fig. 16. Query delay versus update interval.

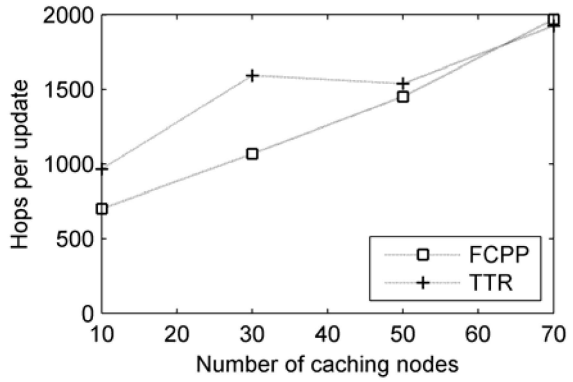

Fig. 17. Traffic overhead per update versus number of caching nodes.

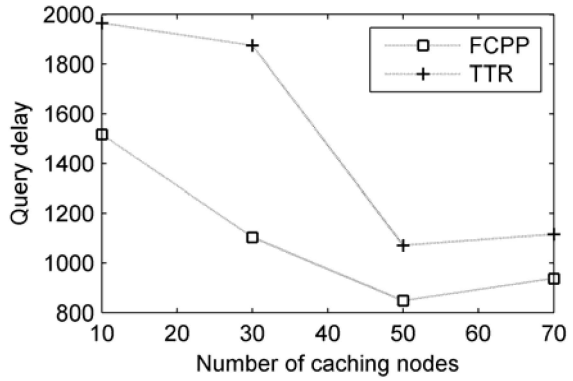

Fig. 18. Query delay versus number of caching nodes.

6.4.2 Dealing with Different Number of Caching Nodes

In this experiment, we increase the number of caching nodes from 10 to 70. We find that FCPP becomes less effective in terms of both traffic overhead and query delay when the number of caching node increases (Figs. 17 and 18). This is mainly because when there are more caching nodes, the traffic overhead for the INV & ACK process greatly increases. Moreover, when the source data are more widely cached, more cache queries can be directly served and the cost-effectiveness of FCPP in dealing with frequent cache queries is greatly affected.

6.4.3 Dealing with Different Node Mobility Levels and Models

In this experiment, we first evaluate the impact of node speed under the random waypoint model. We find that

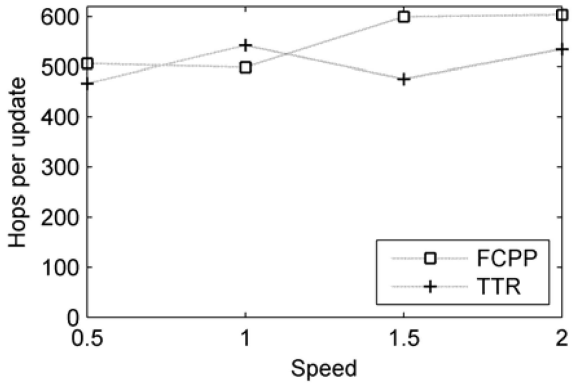

Fig. 19. Traffic overhead per update versus node speed.

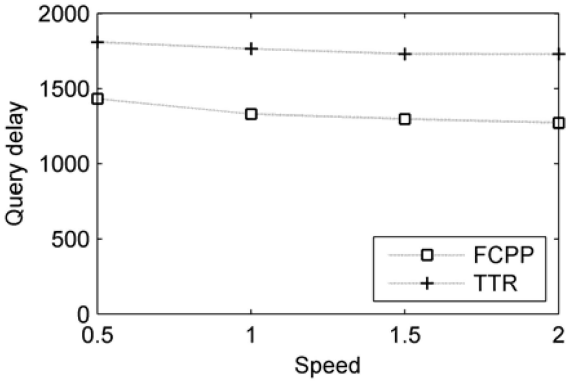

Fig. 20. Query delay versus node speed.

when the mobile hosts move more quickly, the traffic overhead of FCPP slightly increases, as shown in Fig. 19. This is because in more dynamic environments, the data source node needs to set more conservative time-out values. The traffic overhead of Pull with TTR vibrates a little, but does not change much. It outperforms FCPP by around 10 percent when the node speed is increased to 2.0 m/s. Pull with TTR does not require complicated interactions between the data source node and the caching nodes. Thus, it better copes with dynamic environments.

The query delay imposed by both schemes is more stable, as shown in Fig. 20. FCPP outperforms Pull with TTR by around 20 percent. We find that FCPP is more cost-effective in terms of query delay, but is less effective in terms of traffic overhead at the same time. This is because the query delay is mainly decided by how many cache queries can be directly served by the caching nodes. FCPP achieves less query delay since it grants optimized time-out values to the caching nodes. It induces more traffic overhead in some cases, mainly due to the round-trip INV & ACK process.

We also change the mobility model to the Gauss-Markov model (the tuning parameter α [18] is set to 75 percent). We find that FCPP becomes slightly more cost-effective when the mobile hosts follow the Gauss-Markov model. The traffic overhead is reduced by up to 3 percent, while the query delay is decreased by up to 2 percent. This is mainly because under the Gauss-Markov model, mobile hosts change their speed and direction in a more graceful manner, resulting in a less dynamic network environment.

6.5 Lessons Learned

Based on the evaluation results, we conclude that FCPP enables users to efficiently make trade-offs between cache consistency and the consistency maintenance cost. Specifically, FCPP is more accurate when users have more stringent requirements. FCPP is more cost-effective when: 1) faced with frequent cache queries and less frequent data updates; 2) the source data are not widely cached; and 3) the network connection is more stable.

7 Conclusion and Future Work

In this paper, we study the problem of how to satisfy user-specified consistency requirements at the minimum cost. Toward this objective, our contributions are as follows:

1. we proposed a general consistency model PDC, enabling users to flexibly specify their consistency requirements in two orthogonal dimensions;
2. we developed the FCPP algorithm to maintain cache consistency under the PDC model;
3. we derived an analytical model which achieves optimized combination of push and pull in FCPP; and
4. we conducted extensive simulations to demonstrate the cost-effectiveness of FCPP in a variety of computing environments.

In our future work, we will study how to satisfy heterogeneous consistency requirements of different users. We will also study how to enable cooperation among caching nodes to cost-effectively propagate data updates.

Acknowledgments

This work is supported by the UGC of Hong Kong under the CERG grant (PolyU 5105/05E), the National Natural Science Foundation of China (No. 60903024, 60736015, 60721002), the National Grand Fundamental Research 973 Program of China (No. 2009CB320702), and the "Climbing" Program of Jiangsu Province, China (No. BK2008017).

References

[1] L. Yin and G. Cao, "Supporting Cooperative Caching in Ad Hoc Networks," *IEEE Trans. Mobile Computing,* vol. 5, no. 1, pp. 77-89, Jan. 2006.
[2] W. Lau, M. Kumar, and S. Venkatesh, "A Cooperative Cache Architecture in Supporting Caching Multimedia Objects in MANETs," *Proc. Fifth Int'l Workshop Wireless Mobile Multimedia,* 2002.
[3] F. Sailhan and V. Issarny, "Cooperative Caching in Ad Hoc Networks," *Proc. IEEE Int'l Conf. Mobile Data Management (MDM),* 2003.
[4] P. Cao and C. Liu, "Maintaining Strong Cache Consistency in the World Wide Web," *IEEE Trans. Computers,* vol. 47, no. 4, pp. 445-457, Apr. 1998.
[5] S. Lim, W. Lee, G. Cao, and C. Das, "A Novel Caching Scheme for Internet Based Mobile Ad Hoc Networks," *Proc. 12th Int'l Conf. Computer Comm. and Networks (ICCCN),* 2003.
[6] J. Cao, Y. Zhang, L. Xie, and G. Cao, "Data Consistency for Cooperative Caching in Mobile Environments," *Computer,* vol. 40, no. 4, pp. 60-67, Apr. 2007.
[7] S. Zhu and C. Ravishankar, "Stochastic Consistency and Scalable Pull-Based Caching for Erratic Data Stream Sources," *Proc. 30th Very Large Data Bases (VLDB) Conf.,* 2004.
[8] J. Howard et al., "Scale and Performance in a Distributed File System," *ACM Trans. Computer Systems,* vol. 6, no. 1, pp. 51-81, 1988.
[9] C. Gray and D. Cheriton, "Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency," *Proc. ACM Symp. Operating System Principles,* 1989.
[10] V. Duvvuri, P. Shenoy, and R. Tewari, "Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web," *IEEE Trans. Knowledge and Data Eng.,* vol. 15, no. 4, pp. 1266-1276, Sept./Oct. 2003.
[11] B. Urgaonkar, A. Ninan, M. Raunak, P. Shenoy, and K. Ramamritham, "Maintaining Mutual Consistency for Cached Web Objects," *Proc. 21st Int'l Conf. Distributed Computing Systems (ICDCS),* Apr. 2001.
[12] J. Lan, X. Liu, P. Shenoy, and K. Ramamritham, "Consistency Maintenance in Peer-to-Peer File Sharing Networks," *Proc. Third IEEE Workshop Internet Applications,* 2003.
[13] J. Cao, Y. Zhang, L. Xie, and G. Cao, "Consistency of Cooperative Caching in Mobile Peer-to-Peer Systems over MANET," *Int'l J. Parallel, Emergent, and Distributed Systems,* vol. 21, no. 3, pp. 151-168, June 2006.
[14] Y. Huang, J. Cao, and B. Jin, "A Predictive Approach to Achieving Consistency in Cooperative Caching in MANET," *Proc. First Int'l Conf. Scalable Information Systems, P2PIM Workshop Session,* 2006.
[15] M. Corson, J. Macker, and G. Cirincione, "Internet-Based Mobile Ad Hoc Networking," *IEEE Internet Computing,* vol. 3, no. 4, pp. 63-70, July/Aug. 1999.
[16] G. Cao, "Proactive Power-Aware Cache Management for Mobile Computing Systems," *IEEE Trans. Computers,* vol. 51, no. 6, pp. 608-621, June 2002.
[17] M. Bhide, P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy, "Adaptive Push-Pull: Disseminating Dynamic Web Data," *IEEE Trans. Computers,* vol. 51, no. 6, pp. 652-668, June 2002.
[18] T. Camp, J. Boleng, and V. Davies, "A Survey of Mobility Models for Ad Hoc Network Research," *Wireless Comm. and Mobile Computing,* vol. 2, no. 5, pp. 483-502, 2002.
[19] Y. Huang, J. Cao, Z. Wang, B. Jin, and Y. Feng, "Achieving Flexible Cache Consistency for Pervasive Internet Access," *Proc. Fifth Ann. IEEE Int'l Conf. Pervasive Computing and Comm. (PerCom),* pp. 239-250, 2007.
[20] H. Zou, N. Soparkar, and F. Jahanian, "Probabilistic Data Consistency for Wide-Area Applications," *Proc. 16th Int'l Conf. Data Eng.,* 2000.
[21] P. Nuggehalli, V. Srinivasan, and C. Chiasserini, "Energy-Efficient Caching Strategies in Ad Hoc Wireless Networks," *Proc. ACM MobiHoc,* pp. 25-34, 2003.
[22] H. Artail, H. Safa, K. Mershad, Z. Abou-Atme, and N. Sulieman, "COACS: A Cooperative and Adaptive Caching System for Manets," *IEEE Trans. Mobile Computing,* vol. 7, no. 8, pp. 961-977, Aug. 2008.
[23] Y. Saito and M. Shapiro, "Optimistic Replication," *ACM Computing Surveys,* vol. 37, no. 1, pp. 42-81, 2005.
[24] P.A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems.* Addison-Wesley/Longman Publishing Co., Inc., 1986.

Yu Huang received the BSc and PhD degrees in computer science from the University of Science and Technology of China, Hefei, China, in 2002 and 2007, respectively. From 2003 to 2007, he studied in the Institute of Software, Chinese Academy of Sciences, as a coeducated PhD student. He also studied in the Department of Computing, Hong Kong Polytechnic University, as an exchange student from 2005 to 2006. He is currently an assistant professor in the Department of Computer Science and Technology, Nanjing University, China. His research interests include mobile and pervasive computing, software engineering and methodology, and distributed computing. He is a member of the China Computer Federation and the IEEE.

Jiannong Cao received the BSc degree in computer science from Nanjing University, China, in 1982, and the MSc and PhD degrees in computer science from Washington State University, Pullman, in 1986 and 1990, respectively. He is currently a professor in the Department of Computing at Hong Kong Polytechnic University, Hung Hom, Hong Kong, where he is also the director of the Internet and Mobile Computing Lab. Before joining Hong Kong Polytechnic University, he was with the Faculty of Computer Science at James Cook University, University of Adelaide in Australia, and City University of Hong Kong. His research interests include parallel and distributed computing, networking, mobile and wireless computing, fault tolerance, and distributed software architecture. He has published more than 200 technical papers in the above areas. His recent research has focused on mobile and pervasive computing systems, developing testbed, protocols, middleware, and applications. He is a senior member of the China Computer Federation, a senior member of the IEEE, including Computer Society and the IEEE Communication Society, and a member of the ACM. He is also a member of the IEEE Technical Committee on Distributed Processing, IEEE Technical Committee on Parallel Processing, and IEEE Technical Committee on Fault Tolerant Computing. He has served as a member of editorial boards of several international journals, a reviewer for international journals/conference proceedings, and also as an organizing/programme committee member for many international conferences.

Beihong Jin received the BS degree in computer science from Tsinghua University in 1989, and the MS and PhD degrees in computer science from the Institute of Software, Chinese Academy of Sciences, in 1992 and 1999, respectively. Currently, she is a professor in the Institute of Software, Chinese Academy of Sciences. Her research interests include mobile and pervasive computing, middleware, and distributed systems. She has published more than 50 research papers in these areas and holds one China patent. She is a senior member of the China Computer Federation and a member of the ACM.

Xianping Tao received the BSc degree in computer science from the National University of Defense Technology, Changsha, China, and the MSc and PhD degrees in computer science from Nanjing University, China. He is currently a professor in the Department of Computer Science and Technology at Nanjing University. His research interests include software engineering and methodology, middleware systems, and pervasive computing. He is a member of the IEEE.

Jian Lu received the BSc, MSc, and PhD degrees in computer science from Nanjing University, P.R. China. He is currently a professor in the Department of Computer Science and Technology and the director of the State Key Laboratory for Novel Software Technology at Nanjing University. He serves on the Board of the International Institute for Software Technology of the United Nations University (UNU-IIST). He also serves as the director of the Software Engineering Technical Committee of the China Computer Federation. His research interests include software methodologies, software automation, software agents, and middleware systems. He is a member of the IEEE.

Yulin Feng received the PhD degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, in 1982. He visited Stanford University and Carnegie Mellon University as a postdoctoral research scientist during 1982-1985. He was a professor in the Department of Computer Science, University of Science and Technology of China, from 1986 to 1991, and then, he moved to the Institute of Software, Chinese Academy of Sciences, where he is currently a research professor. His research interests are in the areas of Distributed Computing and Systems, Mobile Agents and Computation, System Specifications, and Modeling.