Achieving Flexible Cache Consistency for Pervasive Internet Access

Yu Huang[1,2], Jiannong Cao[1], Zhijun Wang[1], Beihong Jin[2], Yulin Feng[2]
[1]*Internet and Mobile Computing Lab, Dept. of Computing*
Hong Kong Polytechnic Univ., Kowloon, Hong Kong
alberthuang@ustc.edu
{csjcao, cszjwang}@comp.polyu.edu.hk
[2]*Technology Center of Software Engineering, Institute of Software*
Chinese Academy of Sciences, Beijing, China
{jbh, feng}@otcaix.iscas.ac.cn

Abstract

Caching is an important technique to support pervasive Internet access. Cache consistency measures the deviation between the cached data and the source data. In mobile computing environments, especially with ad hoc networks, users are in great need of the flexibility in tuning their consistency requirements, in order to make tradeoffs between the specified cache consistency and the cost incurred. Existing works have used Delta Consistency (DC) and Probabilistic Consistency (PC) which, to some extent, provide the users with such flexibility. In this paper, we propose a general consistency model called Probabilistic Delta Consistency (PDC). PDC covers all existing consistency models including DC and PC, and integrates the flexibility granted by both DC and PC. Thus, PDC enables the users to flexibly specify their consistency requirements in two orthogonal dimensions, namely the deviation in time/value and the ratio of queries gaining the specified consistency. We also propose a consistency maintenance algorithm, called Flexible Combination of Push and Pull (FCPP), which can meet users' consistency requirements specified under the PDC model. An analytical model is derived to achieve the optimized combination of push and pull, so as to ensure the user-specified consistency requirements, while minimizing the consistency maintenance overhead. Extensive simulations are conducted to evaluate the performance of the FCPP algorithm. Evaluation results show that, compared with the widely used Dynamic TTR algorithm, FCPP can save up to 68% of the traffic overhead and reduce the query delay by up to 84%.

1. Introduction

Pervasive network is an important component of the underlying infrastructure of pervasive computing. In recent years, many research efforts have been made to facilitate pervasive access to the Internet through both wired and wireless networks. For example, Internet-based Mobile Ad hoc Networks (IMANETs) [15, 5] have emerged as an approach to providing pervasive Internet access. In an IMANET formed by multiple mobile users, as shown in Fig. 1, the mobile users close to a base station or access point can directly access Internet resources, and hence can serve as gateways to the Internet for other nodes out of the coverage. Therefore, IMANET allows mobile users to access Internet resources either directly or through multi-hop wireless connections. IMANETs can be used in many application scenarios, including university campuses, airports, mobile stores and battlefields to provide Internet access [13].

The limited communication resources (e.g., bandwidth and battery power) and users' mobility make Internet access a challenging task in mobile wireless networks such as IMANETs. One effective and widely-used method to improve system performance is to cache frequently accessed data objects at the data source (gateway node) and a group of caching nodes [1, 2, 3]. Thus, other mobile users can access the cached data objects nearby, with reduced query latency and traffic overhead. In order to ensure valid data access, the cache consistency [6], i.e., consistency among the source data owned by the data source and the cache copies held by a collection of caching nodes, must be maintained properly.

Proceedings of the Fifth Annual IEEE International
Conference on Pervasive Computing and Communications (PerCom'07)
0-7695-2787-6/07 $20.00 © 2007 IEEE

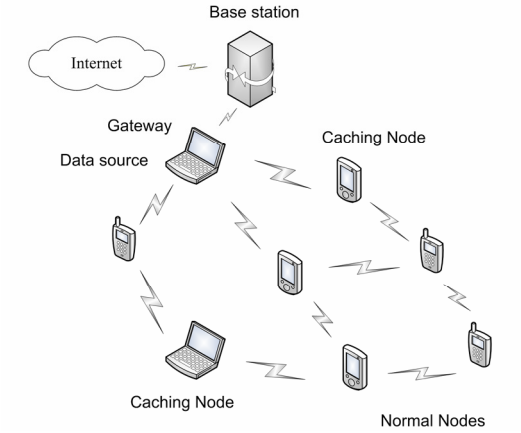

Fig 1. An IMANET for pervasive Internet access

Different types of data objects often have different consistency requirements. For example, suppose a mobile user needs to know the newest stock prices using his smart phone. The accessed stock price should be very up-to-date, such as the price at most 2 minutes ago. The mobile user might also be interested in the weather forecast for the coming day. In this case, the weather information returned can be of several hours ago. Even if the cached data object was not updated in time, the previous version is still useful. On the other hand, in some applications, not all but a certain percentage of the queries are required to satisfy the consistency requirements. For example, a taxi driver may frequently check traffic information. Since the driver can check the traffic information well in advance, he may thus tolerate that certain portion of the accesses is slightly stale. As long as a bounded portion of the accessed information is up-to-date, the driver can efficiently choose the best route.

Therefore, there is a need to develop a consistency model, which allows the users to flexibly tune their consistency requirements for different types of data objects. In doing so, the users can efficiently make tradeoffs between the specified cache consistency and the maintenance cost. In literature, *Delta Consistency* (*DC*) [6] and *Probabilistic Consistency* (*PC*) [7] have been proposed to provide some flexibility for users to tune their consistency requirements either in the maximum acceptable data deviation or in the guaranteed probability of providing valid data.

In this paper, we propose a more general and flexible cache consistency model, called *Probabilistic Delta Consistency* (*PDC*). *PDC* covers all widely studied consistency models including *DC* and *PC*, and allows users to dynamically specify their consistency requirements. In *PDC*, a mobile user can specify their consistency requirements in two orthogonal dimensions, as shown in Fig 2. The dimension along the x-axis specifies the value δ, which denotes the maximum acceptable deviation in time or value between the source data and the cache copy. The dimension along the y-axis specifies the probability p which represents the ratio of queries served by cache copies that must satisfy the specified *DC*. Supporting *PDC* means that the mobile users' queries are served by the cache copies satisfying user-specified *DC* with user-specified probability (formally defined in Section 3). With the *PDC* model, users can flexibly specify their consistency requirements by continuously tuning δ and p.

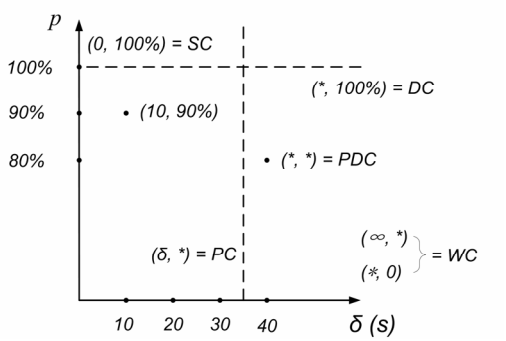

Fig 2. The probabilistic delta consistency model

We also propose an algorithm called *Flexible Combination of Push and Pull* (FCPP) under the *PDC* model. In FCPP, each cache copy is associated with a timeout value which is calculated based on the consistency requirement δ and p. Upon each update, the data source sends an *invalidation* (INV) message to each cache copy that still possesses a valid timeout to invalidate the cache copy. The caching node acknowledges each INV message with an INV_ACK message. The data source can update the source data if it has collected the INV_ACK messages for all the INV messages sent. Otherwise, the data source will not update the data until either the timeout of all the unresponding cache copies expires, or the maximum tolerable delay of the application on the data source is reached. Note that, if the data source can always delay the update until the timeout values on all the unresponding caching nodes expire. FCPP provides *Strong Consistency* (*SC*) and covers the *Lease* protocol [9, 10] as its special case. By adjusting the timeout value associated with the cache copies, FCPP also covers many other cache consistency schemes, such as *Pull each read* [8] and *Invalidation* [4].

An analytical model is derived to calculate the optimized timeout value corresponding to a user-specified consistency requirement, in order to minimize the consistency maintenance overhead. Extensive simulations are conducted to study the

Proceedings of the Fifth Annual IEEE International
Conference on Pervasive Computing and Communications (PerCom'07)
0-7695-2787-6/07 $20.00 © 2007 IEEE

performance of the proposed FCPP algorithm. The evaluation results show that FCPP can satisfy different consistency requirements with high efficiency, in terms of traffic overhead and query delay in a variety of network environments. FCPP can save up to 68% of the traffic overhead and reduce the query delay by up to 84% while satisfying the specified consistency requirement, compared with the widely used *Dynamic TTR* algorithm [11, 12].

The rest of this paper is organized as follows. Section 2 provides an overview of the existing works. Section 3 presents the formal definition of the *PDC* model. In Section 4, we present details in design of the FCPP algorithm. An analytical model is derived to calculate the optimized timeout duration in Section 5. Section 6 presents the experimental evaluation. Finally, Section 7 concludes the paper with a summary and our future works.

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*, the data source and the caching nodes maintain cache consistency, but do not provide any guarantee on the deviation between the source data and the cache copies [6]. To provide consistency requirements between *SC* and *WC*, *Delta consistency* (*DC*) [6] and *Probabilistic Consistency* (*PC*) [7] have been proposed. In *DC*, the users can specify the maximum acceptable deviation between the source data and the cached data, while in *PC*, the users can specify the probability of valid data access. Thus, *DC* and *PC* enable users to specify their consistency requirements in two separate but orthogonal dimensions. The *PDC* model proposed in this paper integrates these two models into a uniform model.

To meet different consistency requirements, many cache consistency maintenance schemes 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*. It is based on the online predictions of data updates and queries. However, these schemes can provide only *SC* or *WC*. Therefore, they are not efficient or applicable in case that the users need to specify various consistency requirements for different types of data objects. A widely used technique to support *DC* is to associate a *Time to Refresh* (TTR) with each cache copy [1, 11, 12, 17]. In order to work efficiently in dynamic environments, the TTR value is dynamically adjusted based on a simple linear model [11, 12]. In [17], the authors proposed the *Push-and-Pull* and the *Push-or-Pull* schemes to provide *DC*. In [13], a *Relay Peer* based scheme was proposed to provide different consistency requirements, including *SC*, *DC* and *WC*. However, none of the above-mentioned schemes can support probabilistic consistency requirements. In [7], the authors proposed an adaptive pull algorithm to support *PC*, but the proposed algorithm is designed only for maintaining the consistency of numerical data values.

So far, to the best of our knowledge, there is no single cache consistency maintenance scheme which can effectively support different user-specified cache consistency requirements, including *SC*, *WC*, *DC*, and *PC*, on various types of data objects. This paper aims to design such a cache consistency maintenance scheme.

3. Probabilistic Delta Consistency

In this section, we first present the formal definition of the *Probabilistic Delta Consistency* (*PDC*) model and then discuss its flexibility and generality.

Let S^t denote the version number of the source data and C^t_j denote the version number of the cached copy on node *j* at time *t*. The version number of the source data is initially set to *zero* and then increased upon each subsequent update. The version number of the cache copy is set to that of the source data at the time when it is synchronized. Then, we can formally define *PDC*(δ, *p*) as follows:

$$\forall t,\ \forall j,\ P\{\exists \tau,\ 0 \leqq \tau \leqq \delta,\ \text{s.t. } S^{t-\tau} = C^t_j\} \geqq p;$$

The mobile users can specify their consistency requirements in two orthogonal dimensions under the *PDC* model. They can specify the data deviation δ and the probability of valid data access *p*. Satisfying the consistency requirement *PDC*(δ, *p*) means that the users' queries are served by cache copies that cannot be stale longer than δ, with a probability no less than *p*.

The users can continuously specify δ in range $[0, \infty)$ and *p* in range [0, 100%], thus obtaining great flexibility in specifying the consistency requirements in the consistency spectrum between *SC* and *WC*. For example, in the scenario discussed in Section 1, a user accesses the stock prices with his smart phone. He can assure that the prices accessed will not be too stale by specifying a small δ (e.g., 20s). Since the user frequently checks the prices, he might be able to tolerate that a small portion of his queries does not satisfy the requirement on δ. He can specify a p value a little less than 100% (e.g., 95%), thus reducing query latency and traffic overhead. In another scenario mentioned in Section 1, a taxi driver frequently

Proceedings of the Fifth Annual IEEE International
Conference on Pervasive Computing and Communications (PerCom'07)
0-7695-2787-6/07 $20.00 © 2007 IEEE

accesses traffic information of the destination to decide which route to choose. If the driver requires that all his queries are served by up-to-date traffic information, the access latency might be intolerable, due to the bandwidth-constrained and unreliable wireless communication. Since the driver can frequently access the traffic information in advance and the traffic information cannot change dramatically, he can specify less stringent requirement on the ratio of valid access p (e.g., 70%). The driver can also relax his requirement on δ (e.g., 90s), in order to reduce the query latency and traffic overhead.

The proposed *PDC* model covers the existing consistency models as special cases: *PDC*(0, 100%) is equivalent to the *SC* model. *PDC*(∞, *) and *PDC*(*, 0) yield the *WC* model. *PDC*(*, 100%) yields the *DC* model. Here, '*' represents any possible values. *Probabilistic Consistency* (*PC*) means that the pre-specified consistency requirement is satisfied with user-specified probability. Thus for specified δ, *PDC*(δ, *) will yield the *PC* model.

4. The Flexible Combination of Push and Pull Algorithm

Details of the proposed *Flexible Combination of Push and Pull* (FCPP) algorithm are presented in this section. In FCPP, we consider a commonly used system model, where each data object is associated with a single node which can update the *source data*. This node is referred to as the *data source*. Each data object can be cached by a group of nodes called *caching nodes*. The data copies held by the caching nodes are called *cache copies*. There are mainly two basic mechanisms for achieving cache consistency: *push* and *pull*. Using *push*, the data source informs every caching node of the data update. Using *pull*, for each data access, a caching node sends a request to the data source to check if the cache copy is up-to-date.

4.1. Design of the FCPP Algorithm

In FCPP, each data object is associated with a timeout duration l. All accesses to a cache copy can be directly served during its valid timeout duration (i.e., $l > 0$). When the timeout expires (l deceases to 0), before serving a data access, the caching node needs to renew its timeout from the data source and to update its own cache copy if the source data has been updated.

On the other hand, the data source maintains the timeout information of all caching nodes. When the source data is ready to be updated, the data source sends an *invalidation* (INV) message to all the cache copies that possess valid timeout values. The caching node acknowledges each INV message with an INV_ACK message. The data source can immediately update the source data only if it receives an INV_ACK message for each INV message sent out. Otherwise, the data source cannot update the source data until either the timeout values of all the unresponding cache copies expire, or the waiting time is over D seconds. Here, D is a system parameter denoting the maximum period the data source can wait before updating the source data. The pseudo code of the FCPP algorithm is presented below.

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 and update the cache copy;

(2.2) After l is renewed, serve the query with the updated cache copy;

Algorithm 2 FCPP on the data source

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) Grant timeout with duration l and the source data to the caching node;

4.2 FCPP as a Generic Scheme

When the data source can always delay the update until the timeout values on all the unresponding caching nodes expire (D can be sufficiently large), FCPP provides *SC* and covers the *Lease* protocol [9, 10] as its special case. When the timeout duration l is set to 0, FCPP becomes the *Pull each read* scheme [8]. When the data source lets l be infinite, FCPP transforms to *Push with ACK*. By adjusting the timeout duration, FCPP flexibly integrates *push* and *pull*.

The timeout duration l is determined by the consistency requirement *PDC* (δ, p). As we will see in the next section, FCPP guarantees *DC* when $l \leqq D+\delta$. In particular, when $l \leqq D$, FCPP provides *SC*. When l

Proceedings of the Fifth Annual IEEE International
Conference on Pervasive Computing and Communications (PerCom'07)
0-7695-2787-6/07 $20.00 © 2007 IEEE

$> D+\delta$, FCPP provides probabilistically guaranteed *DC*. In the next section, we investigate how to decide the timeout duration l to satisfy the user-specified consistency requirement *PDC* (δ, p) while minimizing the consistency maintenance overhead.

5. Analysis of the Optimized Timeout Duration of FCPP

In this section, we derive an analytic model to study the relationship between the timeout duration l and the consistency requirement *PDC* (δ, p), as well as the relationship between l and the traffic overhead. Then we calculate the optimized timeout duration which satisfies the user-specified *PDC* with minimum traffic overhead. In the analysis, we assume that the data update and the query follow *Poisson Process.* The number of hops counted in data transmission is used to measure the traffic overhead. The notations used in the analytical model are listed in Table 1.

Table 1. Notations used in the analytical model

(δ, p)	user-specified consistency requirement
D	maximum time the data source can wait before updating the source data
l	timeout duration in FCPP
N	number of caching nodes
N_k	the k^{th} caching node, k = 1, 2, …, N
$\overline{h}$	average path length between the data source and the caching nodes
w	average frequency of data update
r	average query frequency of a cache copy
t_u	time instant of one source data update

5.1 Providing Probabilistic Delta Consistency

We first argue that when $l \leqq D+\delta$, no stale cache hit occurs in the sense of *Delta Consistency* (*DC*). In *DC*, the users consider the cache copy access be valid if the deviation between the source data and the cache copy is less than δ. Suppose a source data update comes at time t_u, the data source first sends out an INV message to each caching node, and then waits for at most D seconds (Fig 3). In the worst case, every INV message is lost and every caching node has the maximum timeout duration $l = D+\delta$. The data source updates the source data after waiting for D seconds. We find that:

- during period (t_u, $t_u + D$), there is no stale cache hit, since the data source has not updated the source data yet;

- during period ($t_u + D$, $t_u + D+\delta$), no stale cache hit occurs, since the deviation between the source data and the cache copy is bounded by δ. Given the consistency requirement (δ, p) specified under the *PDC* model, the cache copies are still considered to be valid.

Thus, we prove that there is no stale cache hit if $l \leqq D+\delta$.

In case that $l > D+\delta$, in period (t_u, $t_u+D+\delta$), there is no stale cache hit, as discussed above. During period ($t_u+D+\delta$, t_u+l), for a caching node with valid timeout, its original timeout duration x lies in range [$D+\delta$, l]. If this caching node misses the INV message and still serves queries, the expected number of stale cache hits is $(x-D-\delta)r$. Since x is uniformly distributed in range ($D+\delta$, l), the expected number of stale cache hits in (t_u, t_u+l) is:

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

When the data source misses an INV_ACK message from a caching node, two cases may happen. One is that the caching node misses the INV message, and the other is that the caching node has successfully received the INV message, but the corresponding INV_ACK message is lost. We discuss each case as follows:

- in the first case, the data source cannot receive the corresponding INV_ACK message. The expected number of stale cache hits introduced by INV message loss is counted in inequality (1) below;

- in the second case, the data source will falsely decide that the caching node introduces stale cache hits. We take these false stale hits as real stale hits in our analytical model to get the upper bound of the probability of stale hits, in order to ensure the user-specified consistency requirement (δ, p). Our estimation of the number of stale cache hits in this case is conservative.

According to the discussion above, we have that $S \leq \frac{r}{2l}(l-\delta-D)^2$ in both cases.

Proceedings of the Fifth Annual IEEE International
Conference on Pervasive Computing and Communications (PerCom'07)
0-7695-2787-6/07 $20.00 © 2007 IEEE

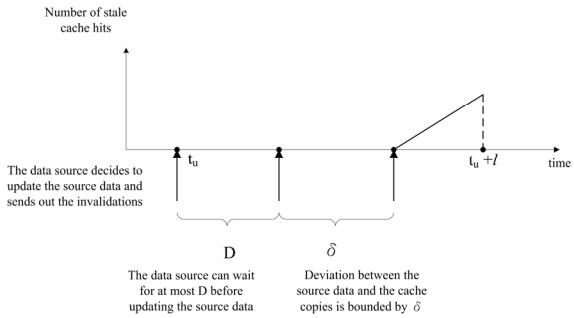

Fig 3.Number of stale cache hits

Based on the estimated number of stale cache hits on each caching node, we find that the total number of stale cache hits is bounded by:

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

Here, for each caching node N_k, $1 \le k \le N$, $I(k)$ has the value *one*, if N_k has a timeout l greater than $D+\delta$ and the INV_ACK message from N_k is missed. Otherwise, $I(k)$ is set to *zero*. $I(k)=1$ indicates that N_k may introduce some stale hits. If N_k has a timeout l less than $D+\delta$, or the data source successfully receives the INV_ACK message from N_k, no stale hits will be induced by this node as discussed above. Hence, we set $I(k)$ to *zero* to eliminate the false stale hits from N_k. When the data source is ready to update the data, it can determine the value of $I(k)$ for each caching node N_k based on l and δ. In inequality (1), p_k denotes the probability that the INV_ACK message from N_k is missed.

Upon each source data update, the data source sets the timeout duration to l. The average interval between the current and the forthcoming source data update is $1/w$. The expected number of cache queries in this period is $r \cdot N / w$ and the expected number of stale cache hits incurred is ES. Thus, we obtain that the probability of DC provided to the users is: $1 - (w \cdot ES)/(r \cdot N)$. Since the probability of DC should be no less than the user-specified probability p, we have: $p \le 1 - (w \cdot ES)/(r \cdot N)$. Based on inequality (1), we can ensure the consistency requirement (δ, p) if we let:

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

To solve this inequality, we find that $d(l-D-\delta)^2 / dl > 0$ 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 then obtain that the value of l is bounded by:

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

5.2 Traffic Overhead

According to the FCPP algorithm, there are mainly two types of traffic overhead: traffic for timeout renewal and that for the INV & ACK process. As shown in Fig 4, when a query comes after the timeout expired, the caching node first renews the timeout value to l and then directly serves the next $\lfloor l \cdot r \rfloor$ (on average) queries. Thus, for every $\lfloor l \cdot r \rfloor + 1$ queries, there will be one timeout renewal on average. So the expected cost for timeout renewal per unit time is: $2 \cdot \bar{h} \cdot N \cdot r / (\lfloor l \cdot r \rfloor + 1)$.

As for data updates, for every $\lfloor l \cdot r \rfloor + 1$ queries on a caching node, $\lfloor l \cdot r \rfloor$ of them occur within valid timeout. So the probability that a caching node has valid timeout is: $\lfloor l \cdot r \rfloor / (\lfloor l \cdot r \rfloor + 1)$. Thus the expected traffic overhead per unit time associated with the INV & ACK process is: $2 \cdot \bar{h} \cdot N \cdot w \cdot \lfloor l \cdot r \rfloor / (\lfloor l \cdot r \rfloor + 1)$. Thus, we obtain the total traffic overhead per unit time:

$$C = 2N \cdot \bar{h} \frac{r + w \lfloor l \cdot r \rfloor}{\lfloor l \cdot r \rfloor + 1} \approx 2r \cdot N \cdot \bar{h} \frac{1 + w \cdot l}{l \cdot r + 1}.$$

In order to study how the timeout duration l affects the traffic overhead, we take the derivation:

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

We find that, in case $w < r$, in order to minimize the traffic overhead, we should increase the timeout duration l as much as possible[1]. Based on the upper bounded of l in inequality (2), the optimal timeout duration l is:

[1] In case $w > r$, the timeout duration should be reduced as much as possible. This result is reasonable since the round-trip INV & ACK process imposes great traffic overhead in case we have more frequent updates than queries. When l is set to zero, FCPP transforms to the traditional Pull scheme and we omit the detailed discussion on this case in this paper.

$$l = (D+\delta) + \frac{(1-p)N}{wA} + \sqrt{\frac{(1-p)^2N^2}{wA^2} + \frac{2(D+\delta)(1-p)N}{wA}} \quad (3)$$

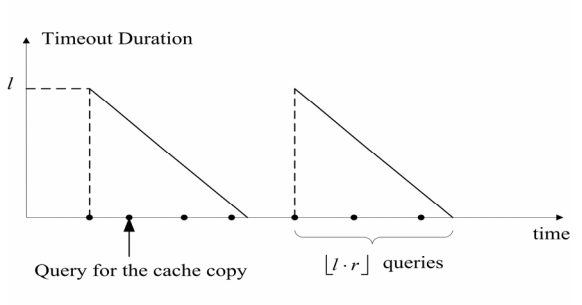

Fig 4. Number of queries within the timeout duration

When we apply our analytical model to cache consistency maintenance, the system parameter D is initialized upon deployment of the caching system and the users specify their consistency requirement $PDC(\delta, p)$ for the cached data objects. Then, based on equation (3), the data source can decide the optimized timeout duration l for each data object, in order to provide user-specified *PDC* while minimizing the traffic overhead.

6. Experimental Evaluations

In this section, we conduct simulations to evaluate the performance of FCPP. We first present the experimental methodology, and then discuss the evaluation results.

6.1 Experiment Methodology

In the experimental evaluation, we first investigate whether FCPP can effectively satisfy different consistency requirements $PDC(\delta, p)$. We then study the performance of FCPP by varying the data update interval and the number of caching nodes. We choose to vary the consistency requirement (δ, p), the update interval and the number of caching nodes because these parameters have the most significant impacts on the performance of FCPP according to the analytical result in equation (3). We study the cost-effectiveness of FCPP based on extensive performance comparison with the *Pull Algorithm with Dynamic TTR* (named as DynTTR in short) [11, 12]. Note that, DynTTR cannot locally guarantee user-specified *PDC*. We centrally configure it to provide specified consistency requirements to conduct the performance comparison here. The following performance metrics are used in the performance comparisons:

- *(δ, p)*: with probability p, deviation between the source data and the cache copy is bounded by δ;

- *Traffic overhead*: average number of hops counted of consistency maintenance message propagation;

- *Query delay*: average delay imposed due to propagating the consistency maintenance messages;

Detailed experimental configurations are listed in Table 2.

Table 2. Experimental configurations

Network area	200×200 m^2
Size of network	80
Transmission range	15
Mobility model	*Random way point* [18]
Average speed	0.5 m/s
Patter of date updates and queries	*Poisson process*
Maximum portion of crashed nodes	10%
Probability of message loss per hop	5%
Maximum delay before data update	2 s
Average interval between queries	5 s

The default values of consistency requirement, average update interval and number of caching nodes are set as follows:

- $(\delta, p) = (5s, 90\%)$;

- average update interval: 20s;

- number of caching nodes: 10;

We will study the impacts of varying these parameters in the following experiments.

6.2 Effects of Tuning the Consistency Requirement (δ, p)

In this experiment, we relax the initial consistency requirement in two orthogonal dimensions. We first decrease p from 90% to 80%, 70% and 60%, and then increase δ from 5 to 10, 15 and 20 seconds. Note that, parameters δ and D are bound together, as shown in equation (3). Thus, tuning δ has the same effect as tuning D. Our first observation is that the FCPP algorithm can locally provide various user-specified consistency requirements, as shown in Figs. 5 and 8. We also find that in most cases, the actual probability of *DC* provided to the users is slightly more than the user-specified probability. Specifically, as shown in Fig 8, the actual probability is no more than 95%, while the user-specified probability is 90%. But also note that, when the user-specified probability p is

decreased to 70% and 60%, the discrepancy between the actual probability provided to the users and the specified probability goes up to 15% (Fig. 5), which shows that FCPP is less accurate in estimating the probability of *DC* if the specified probability *p* is low. It is mainly because the caching nodes get longer timeout duration *l* when the probability *p* is decreased. Thus, there are more chances for the data source to falsely calculate the expected number of stale cache hits. This will make the estimation of stale cache hits more conservative, as discussed in Section 5.1.

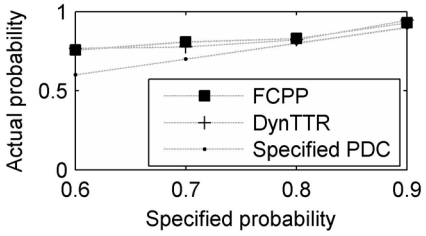

Fig 5. Probability of *DC* vs. *p*

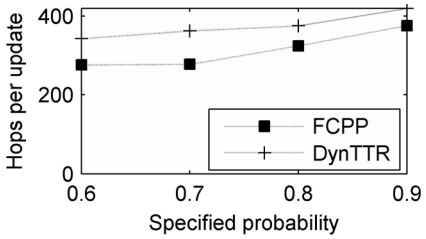

Fig 6. Traffic overhead per update vs. *p*

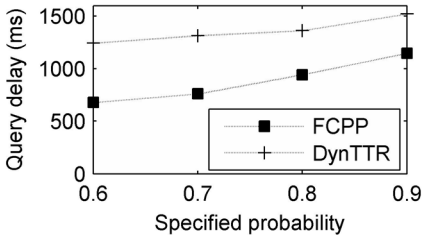

Fig 7. Query delay vs. *p*

We also find that, when the consistency requirement is relaxed in two different dimensions, the performance of FCPP changes differently. FCPP becomes more cost-effective in terms of both traffic overhead and query delay (Figs. 6 and 7) when the probability *p* is decreased. The traffic overhead saved increases from 11% (p = 90%) to 31% (p = 70%). The query delay saved increases from 32.83% (p = 90%) to 83.65% (p = 60%). Here, the percentage of traffic overhead or query delay saved refers to the ratio of the saved cost to the cost imposed by FCPP. FCPP becomes less effective when δ is increased (Figs. 9 and 10). The traffic overhead saved decreases from 24% (δ = 10) to 4% (δ = 20) and the query delay saved decreases from 64% (δ = 10) to 57% (δ = 20). This is mainly because, when the users have less stringent consistency requirements on *p*, the caching nodes can get longer timeout duration from the data source, according to equation (3). Thus, more cache queries can be directly served in FCPP. Meanwhile, when users have less stringent consistency requirements on δ, the simple linear model used in DynTTR has higher probability to gradually tune the TTR and meet users' consistency requirements with less cost. Thus, FCPP becomes comparatively less cost-effective. Overall, the evaluation results show that the FCPP algorithm is more effective in dealing with more strict requirements on δ and looser requirements on *p*, compared with the DynTTR algorithm.

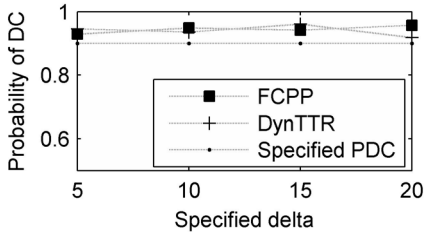

Fig 8. Probability of *DC* vs. δ

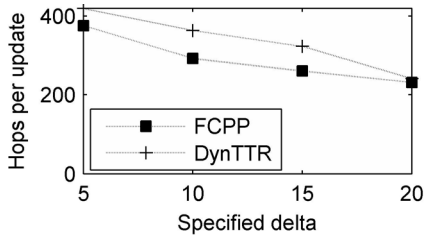

Fig 9. Traffic overhead per update vs. δ

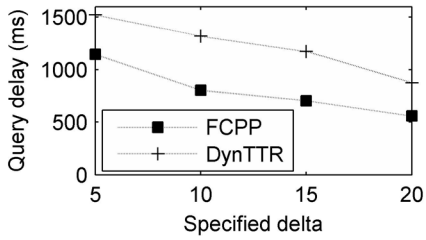

Fig 10. Query delay vs. δ

6.3 Effects of Tuning the Update Frequency

In this experiment, we increase the average interval between data updates from 20 to 80 seconds. We first find that the total traffic overhead decreases as the number of updates decreases (Fig 12). We also find that FCPP becomes much more effective in terms of both traffic overhead per update and query delay, as shown in Figs 13 and 14. The traffic overhead saved increases from 12% (interval = 20) to 64% (interval = 80), and the query delay saved increases from 33%

Proceedings of the Fifth Annual IEEE International
Conference on Pervasive Computing and Communications (PerCom'07)
0-7695-2787-6/07 $20.00 © 2007 IEEE

(interval = 20) to 70% (interval = 80). For lower update frequencies, the data source is expected to set longer timeout duration for the caching nodes and more queries can be directly served, which reduces both traffic overhead and query delay, whereas DynTTR cannot effectively adapt to the changes in data updates. Note that when the update frequency is lower while the query frequency holds, more queries are associated with one update. Hence, the number of hops per update increases, as show in Fig. 13. In addition, the total cost for consistency maintenance decreases, as shown in Fig. 12.

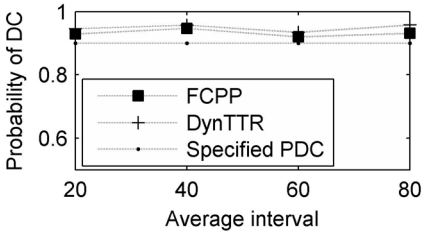

Fig 11. Probability of *DC* vs. update interval

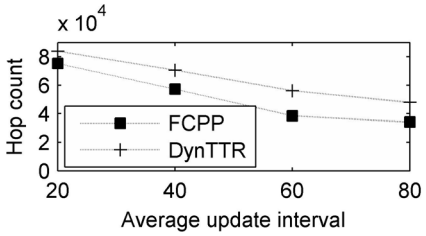

Fig 12. Total traffic overhead vs. update interval

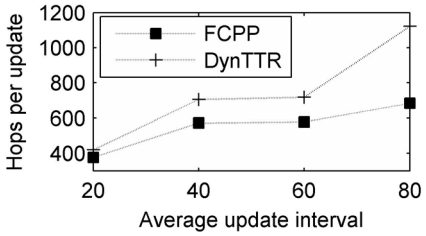

Fig 13. Traffic overhead per update vs. update interval

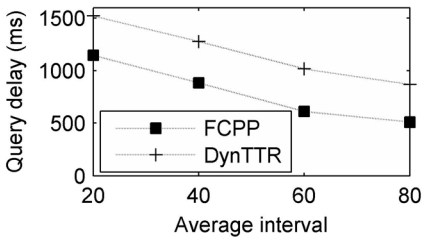

Fig 14. Query delay vs. update interval

6.4 Effects of Tuning the Number of Caching Nodes

In this experiment, we increase the number of caching nodes from 10 to 30, 50 and 70. We find that FCPP becomes less effective in terms of both traffic overhead and query delay when the source data is more widely cached, as shown in Figs 16 and 17. The traffic overhead saved decreases from 67.58% (# of caching nodes = 10) to 5% (# of caching nodes = 70), while the query delay saved decreases from 70% (# of caching nodes = 30) to 42% (# of caching nodes = 70). This is caused by the round-trip traffic overhead and query latency in the INV&ACK process of FCPP.

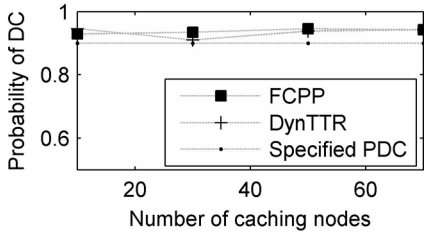

Fig 15. Probability of *DC* vs. number of caching nodes

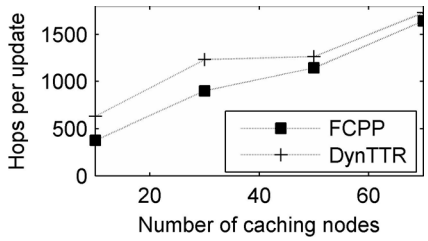

Fig 16. Traffic overhead per update vs. number of caching nodes

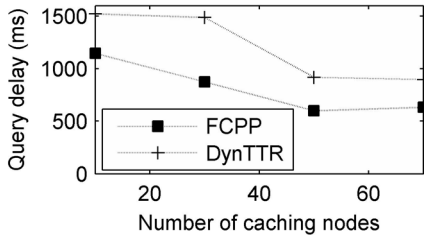

Fig 17. Query delay vs. number of caching nodes

Based on the evaluation results, we conclude that FCPP effectively provides user-specified *PDC* in a variety of network environments. In particular, FCPP is significantly more effective than DynTTR when 1) the users have looser consistency requirements on probability p, or more strict requirements on δ, or 2) there are less frequent data updates and more frequent queries, or 3) the source data is not quite widely cached.

7. Conclusion and Future Work

In this paper, we have addressed the problem of how to provide the users with flexibility in specifying their consistency requirements, and how to satisfy the flexible user-specified consistency requirements with

Proceedings of the Fifth Annual IEEE International
Conference on Pervasive Computing and Communications (PerCom'07)
0-7695-2787-6/07 $20.00 © 2007 IEEE

minimum overhead. Toward this objective, our contributions can be described as follows: (1) we have proposed a general consistency model *PDC*, allowing users to flexibly specify their consistency requirements in two orthogonal dimensions; (2) we have developed the FCPP algorithm to maintain cache consistency under the *PDC* consistency model. FCPP flexibly and efficiently combines push and pull based on timeouts; (3) we have derived an analytical model for FCPP to calculate the optimized timeout duration, so as to provide user-specified *PDC* with minimum traffic overhead in IMANETs; (4) we have conducted comprehensive experimental simulations to evaluate the performance of FCPP, by comparing it with the DynTTR algorithm.

In our future work, we will study how to enable efficient cooperation among the caching nodes, in order to further reduce the cache maintenance cost. We also plan to study how to satisfy heterogeneous consistency requirements of the users.

Acknowledgements

This work is supported by the UGC of Hong Kong under the CERG grant PolyU 5105/05E and the National Natural Science Foundation of China under Grant No. 60673123.

References

[1] G. Cao, L. Yin, and C. Das, Cooperative Cache-Based Data Access in Ad Hoc Networks, IEEE Computer, pp. 32-39, Feb. 2004.

[2] W. Lau, M. Kumar and S. Venkatesh, A Cooperative Cache Architecture in Supporting Caching Multimedia Objects in MANETs, Proc. Fifth Intl Workshop Wireless Mobile Multimedia, 2002.

[3] F. Sailhan and V. Issarny, Cooperative Caching in Ad hoc Networks, IEEE International Conference on Mobile Data Management (MDM), 2003.

[4] P. Cao, C. Liu, Maintaining Strong Cache Consistency in the World Wide Web, IEEE. Trans. on Computers, Vol. 47, No. 4, 1998.

[5] S. Lim, W. Lee, G. Cao and C. Das, A Novel Caching Scheme for Internet based Mobile Ad Hoc Networks, in proc. 12th Intl. Conf. on Computer Communications and Networks (ICCCN), 2003.

[6] J. Cao, Y. Zhang, L. Xie and G. Cao, Data Consistency for Cooperative Caching in Mobile Environments, to appear in IEEE Computer.

[7] S. Zhu and C. Ravishankar, Stochastic Consistency and Scalable Pull-Based Caching for Erratic Data Stream Sources, in Proc. Of the 30th VLDB Conf., 2004.

[8] J. Howard, M. Kazar, et al, Scale and Performance in a Distributed file System, ACM Trans. on Computer Systems, Vol. 6, No. 1, 1988.

[9] C. Gray and D. Cheriton, Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency, ACM Symposium on 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. on Knowledge and Data Engineering, Vol. 15, No. 4, 2003.

[11] B. Urgaonkar, A. Ninan, M. Raunak, P. Shenoy and K. Ramamritham, Maintaining Mutual Consistency for Cached Web Objects, In Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS), Phoenix, AZ, April 2001.

[12] J. Lan, X. Liu, P. Shenoy and K. Ramamritham, Consistency Maintenance in Peer-to-Peer File Sharing Networks, the 3rd IEEE Workshop on 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, Intl. J. of Parallel, Emergent, and Distributed Systems, Vol. 21, No. 3, June 2006.

[14] Y. Huang, J. Cao and B. Jin, A Predictive Approach to Achieving Consistency in Cooperative Caching in MANET, in Proc. of the 1st Intl. Conf. on Scalable Information Systems, P2PIM workshop session, ACM Press, New York, USA, 2006.

[15] M. Corson, J. Macker and G. Cirincione, Internet-based Mobile Ad Hoc Networking, in IEEE Internet Computing, pp.63-70, July-August, 1999.

[16] G. Cao, Proactive Power-aware Cache Management for Mobile Computing Systems, IEEE Trans. on Computers, Vol51, No. 6, 2002.

[17] M. Bhide, P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham and Prashant Shenoy, Adaptive Push-Pull: Disseminating Dynamic Web Data, IEEE. Trans. on Computers, Vol. 51, No. 6, June 2002.

[18] T. Camp, J. Boleng, and V. Davies, A Survey of Mobility Models for Ad Hoc Network Research, Wireless Communications & Mobile Computing (WCMC), Vol. 2, No. 5, 2002.