A Selective Push Algorithm for Cooperative Cache Consistency Maintenance over MANETs

Yu Huang[1,2], Beihong Jin[3], Jiannong Cao[4], Guangzhong Sun[5], Yulin Feng[3]

[1]State Key Laboratory for Novel Software Technology (Nanjing Univ.), Nanjing, China
[2]Dept. of Computer Science and Technology, Nanjing Univ., Nanjing, China
yuhuang@ics.nju.edu.cn

[3]Technology Center of Software Engineering, Institute of Software
Chinese Academy of Sciences, Beijing, China
{jbh, feng}@otcaix.iscas.ac.cn

[4]Internet and Mobile Computing Lab, Dept. Of Computing
Hong Kong Polytechnic Univ, Kowloon, Hong Kong
csjcao@comp.polyu.edu.hk

[5]Dept. of Computer Science and Technology
Univ. of Science and Technology of China, Hefei, China
gzsun@ustc.edu.cn

Abstract. Cooperative caching is an important technique to support efficient data dissemination and sharing in Mobile Ad hoc Networks (MANETs). In order to ensure valid data access, the cache consistency must be maintained properly. Many existing cache consistency maintenance algorithms are stateless, in which the data source node is unaware of the cache status at each caching node. Even though stateless algorithms do not pay the cost for cache status maintenance, they mainly rely on broadcast mechanisms to propagate the data updates, thus lacking cost-effectiveness and scalability. Besides stateless algorithms, stateful algorithms can significantly reduce the consistency maintenance cost by maintaining status of the cached data and selectively propagating the data updates. Stateful algorithms are more effective in MANETs, mainly due to the bandwidth-constrained, unstable and multi-hop wireless communication. In this paper, we propose a stateful cache consistency maintenance algorithm called *Greedy Walk-based Selective Push* (GWSP). In GWSP, the data source node maintains the Time-to-Refresh value and the cache query rate associated with each cache copy. Thus, the data source node propagates the source data update only to caching nodes which are in great need of the update. After recipients of the source data update have been decided, GWSP employs a greedy but efficient strategy to propagate the update among the selected caching nodes. Extensive simulations are conducted to evaluate the performance of GWSP. The evaluation results show that, compared with the widely used *Pull with Dynamic TTR* algorithm, GWSP can save up to 41% traffic overhead and reduce the query latency by up to 85% for cache consistency maintenance in cooperative caching over MANETs.

Keywords: Mobile Ad hoc Networks, Cooperative Caching, Cache Consistency, Stateful, Cache Status Maintenance, Selective Push.

1 Introduction

Mobile Ad hoc Networks (MANETs) have received considerable attention due to the potential applications in pervasive Internet access, outdoor assemblies and disaster salvage [1, 2, 3, 20]. Ad hoc networks feature in their quick deployment and easy reconfiguration, which makes them ideal in situations where installing an infrastructure is too expensive or too vulnerable. The primary goal of deploying MANETs is to support pervasive and efficient data dissemination and sharing [2, 3, 20]. For example, in a MANET shown in Fig. 1, the mobile hosts close to an access point can directly access the Internet, and hence can serve as gateway nodes. Other mobile hosts access Internet resources via the gateway nodes through multi-hop wireless connections. In another example, several commanding officers and a group of soldiers form a MANET in fulfilling a mission of disaster salvage. The commanding officers can access useful information, such as the geographic information, via the satellites. The useful information and commands from the officers can then be efficiently disseminated via the MANET.

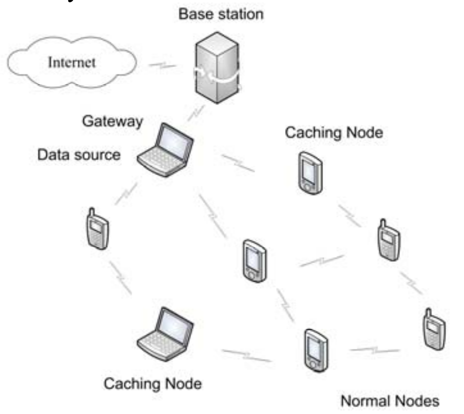


Fig. 1. A MANET with one data source node and multiple caching nodes

However, the limited communication resources (e.g., bandwidth and battery power) and users' mobility make pervasive data dissemination and sharing a challenging task in MANETs. One effective and widely-used method to improve the performance of data access in MANETs is to use *Cooperative Caching*, i.e., to cache frequently accessed data objects at the *data source node* (gateway node) and a group of *caching nodes* [3, 4, 5, 13, 20]. Thus, other mobile users can access the cached data objects nearby, with reduced traffic overhead and query latency.

In order to ensure valid data access, the cache consistency [2, 13, 20, 21], i.e., consistency among the source data owned by the data source node and the cache copies held by the caching nodes, must be maintained properly. Cache consistency maintenance algorithms can be divided into two main categories: *stateful* [17] and *stateless* [15, 16], based on whether the cache status is maintained on the data source node [21]. Existing consistency maintenance algorithms for MANET (e.g., [2, 13, 20]) are mainly stateless, in which the data source node is unaware of status of the cached data. The data source node mainly relies on broadcast mechanisms to propagate the data updates, which inevitably introduces much redundant data update propagation.

Besides stateless algorithms, there also exist stateful consistency maintenance algorithms in the literature. In stateful algorithms, the data source node maintains the cache status (e.g. the Time-to-Refresh value [9]) of each cache copy. Based on the cache status maintained, the data source node can hence selectively propagate the data updates to the caching nodes which are in great need of the updates, i.e., caching nodes whose cache copy will soon expire. Compared with stateless algorithms, stateful ones significantly reduce the consistency maintenance cost by selectively propagate the data updates based on the cache status. We argue that stateful algorithms are more suitable for MANETs, since the power consumption of data transmission (for data update propagation) is much more than that of local computation (for cache status maintenance) on wireless terminals [10]. So far, to the best of our knowledge, no stateful cache consistency maintenance algorithm has been proposed for MANETs.

In this paper, we propose a stateful cache consistency maintenance algorithm called *Greedy Walk-based Selective Push* (GWSP). In GWSP, the data source node maintains the *Time to Refresh* value and the cache query rate associated with each cache copy. Based on the cache status, the data source node can hence make online decisions on which cache copies are in need of the data updates upon each update. When recipients of the data update have been decided, GWSP employs the *greedy walk* mechanism to propagate the data update among the selected caching nodes. Cooperation among the data source node and caching nodes amortizes the cost for data update propagation.

Extensive simulations are conducted to evaluate the performance of GWSP. The evaluation results show that, compared with the stateless *Pull with Dynamic TTR* algorithm, GWSP can save up to 41% traffic overhead and reduce the query latency by up to 85%.

The rest of this paper is organized as follows. Section 2 presents design and analysis of the GWSP algorithm. In Section 3, we present the experimental evaluation. Finally, Section 4 concludes the paper with a summary and the future work.

2 Greedy Walk-based Selective Push

The *Greedy Walk-based Selective Push* (GWSP) algorithm adopts a widely accepted system model [2, 13, 20], in which each data object is associated with a single node that can update the *source data*. This node is referred to as the *data source node*. Each data object can be cached by a collection of nodes called the *caching nodes*. The data copies held by the caching nodes are called the *cache copies*[1]. There are two basic mechanisms for cache consistency maintenance: *push* and *pull*. Using push, the data source node informs the caching nodes of data updates. Using pull, the caching node sends a request to the data source node to check the update.

In designing GWSP, we assume that the source data updates and the cache queries follow the *Poisson Process* of rate λ_u and λ_q respectively [22]. We also assume that

[1] In this paper, we do not consider the issue of caching node membership. This issue should be addressed by the cooperative caching mechanism, based on which we further focus on the issue of cache consistency maintenance.

the routing protocol employed in the network layer provides the hop count between each pair of nodes, and the hop count of data transmission is used to measure the consistency maintenance cost [3].

2.1 Providing Delta Consistency by Pull with TTR

GWSP provides Delta Consistency based on the *Pull with TTR algorithm*. In *Pull with TTR*, each cache copy is associated with a timeout value *Time to Refresh* (TTR). The initial value of TTR is set to δ. When TTR is valid (TTR > 0), the caching node can directly serve cache queries. When TTR expires, the caching node first pulls the data source node to update the cache copy and to renew TTR to δ. Then the caching node can directly serve cache queries. By associating a TTR value with each cache copy, GWSP guarantees that deviation between the source data and the cache copy will not be over δ, thus ensuring Delta Consistency [2, 21].

Although the *Pull with TTR* algorithm guarantees Delta Consistency, it is not cost-effective, mainly due to the round–trip consistency maintenance cost imposed by the pull mechanism. Therefore, GWSP employs a stateful selective push mechanism to save consistency maintenance cost.

2.2 Selective Push

Using push, the data source node informs the caching nodes of data updates, which only imposes one-way consistency maintenance cost (traffic overhead, query latency etc.). However, if the data source node is unaware of the cache status of each cache copy, there exist redundant data update propagations in the following two cases:

- After the cache copy and the associated TTR are renewed via push, there comes no cache query before the TTR expires;
- Before the TTR expires, there are multiple data updates (only the last data update should make the data source node push the caching nodes);

Thus, the following design principles should be followed in designing GWSP: use the push mechanism to save consistency maintenance cost, but

- Push only if the cache copy is expected to serve queries;
- Push if there are probably no other data updates before TTR expires;

The detailed design of the GWSP algorithm is. Upon the source data update at t_u, we consider the cache status on one caching node, as shown in Fig. 2. The TTR of this cache copy was renewed (via push or pull) at time t_0. Thus, the remaining TTR of this caching node at time t_u is $TTR_{remain} = t_0+\delta-t_u$, and the TTR renewed is $TTR_{renew} = t_u - t_0$. According to the design principles, the data source node pushes one caching node only when the following two requirements are both satisfied:

1. The probability that there is at least one query in period $(t_0+\delta, t_u+\delta)$ is greater than threshold value τ_q;
2. The probability that there is at least one update in period $(t_u, t_0+\delta)$ is less than threshold value τ_u;

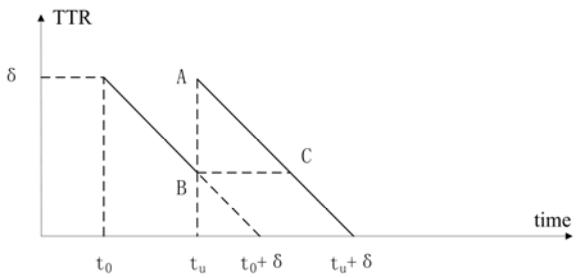


Fig. 2. Cache status on one caching node.

For a *Poisson Process* with rate λ, the probability that there are k events in a period of length t is: $P\{t, k\} = (\lambda t)^k e^{-\lambda t} / k!$. Thus, the probability that there is at least one query in period $(t_0+\delta, t_u+\delta)$ is:

$$1 - P\{(t_u+\delta)-(t_0+\delta), 0\} = 1 - P\{t_u - t_0, 0\} = 1 - e^{-\lambda_q (t_u - t_0)}$$

Thus, for *Requirement (1)*, we have that: $1 - e^{-\lambda_q (t_u - t_0)} \geq \tau_q$, which is equivalent to:

$$TTR_{renew} = t_u - t_0 \geq -\log(1 - \tau_q) / \lambda_q \qquad (1)$$

Note that each caching node maintains the query rate λ_q for the cache copy. When the caching node pulls the data source node, it will piggyback the value λ_q in the pull message. Thus, the data source node obtains the query rate λ_q of each cache copy. The data source node also maintains the TTR value of each cache copy.

The probability that there is at least one update in period $(t_u, t_0+\delta)$ is:

$$1-P\{t_0+\delta - t_u, 0\} = 1 - e^{-\lambda_u (t_0 + \delta - t_u)}$$

Thus, for *Requirement (2)*, we have that $1 - e^{-\lambda_u (t_0 + \delta - t_u)} \leq \tau_u$, which is equivalent to:

$$TTR_{remain} = t_0 + \delta - t_u \leq -\log(1 - \tau_u) / \lambda_u \qquad (2)$$

Based on inequality (1) and (2), the data source node can make on-line decisions on which caching nodes should receive the PUSH message upon each data update.

2.3 Greedy Walk-based Data Update Propagation

After the data source node has decided the *Push Set*, i.e., the caching nodes which should receive the data update, it needs to decide how to propagate the data updates among the selected caching nodes. GWSP employs a greedy but efficient strategy *Greedy Walk* to disseminate the PUSH message (which contains the source data update and IDs of the push set nodes) to all caching nodes in the push set. The data source node sends the PUSH message to the nearest caching node in the push set via unicast. The caching node which receives the PUSH message deletes itself from the push set. It acknowledges the PUSH message with a PUSH_ACK message, and then goes on relaying the push message to the nearest caching node in the push set (The path length between two nodes are provided by the routing protocol, according to our assumption). This process is repeated until the push set is empty. The last caching node will send a PUSH message to the data source node, which initiates the PUSH message propagation process.

If the sender of some PUSH message does not receive the corresponding PUSH_ACK (since either the PUSH message or the PUSH_ACK message is lost), it will wait for at most τ_{ACK} seconds. Then, it will send the PUSH message to the data source node. We require that PUSH messages should be acknowledged, in order to deal with the message loss, which often occurs in MANETs.

When the data source node receives the PUSH_ACK message, it obtains the time span of the PUSH message propagation process. It also knows the caching nodes which have received the PUSH message. The data source node then updates the TTR values of the cache copies as follows. Let t_{push} denote the time span of the PUSH message propagation process. The data source node sets the TTR values of all cache copies which receives the PUSH message to δ - t_{push}. Hence, the data source node can maintain the TTR values of each cache copy more accurately, by taking into account the time delay imposed by the PUSH message propagation process. The pseudo-code of the GWSP algorithm is listed below.

```
Algorithm 1 GWSP on the data source node
Upon each source data update
// deciding the Push Set
(1) Put all caching nodes satisfying inequality (1) and (2) into the push set;
// propagating the source data updates by Greedy Walk
(2) Send the source data update and the push set information in a PUSH
message to the nearest caching node in the push set;

Upon receiving PUSH message
(3) Decide all the caching nodes which have received the PUSH message and
the time span of the push process $t_{push}$;
(4) Update the TTR of the caching nodes which received the PUSH message
by letting $TTR = \delta - t_{push}$;
```

```
Algorithm 2 GWSP on a caching node
Upon receiving a cache query
(1) If (TTR = 0) pull the data source node to update the cache copy and TTR;
(2) Serve the cache query;

Upon receiving a PUSH message
(3) Reply the PUSH message with PUSH_ACK;
(4) Delete itself from the push set;
(5) If( size of the push set > 0) relay the PUSH message to the nearest
caching node in the push set;
(6) Else send the PUSH message to the data source node;

Upon receiving a PUSH_ACK message
(7) If(no PUSH_ACK is received within time $\tau_{ACK}$)
send the PUSH message to the data source node;
```

2.4 Analysis of the Greedy Walk Strategy

Finding the optimal path for propagating the PUSH message can be easily proven to be an instance of the *Traveling Salesman Problem* (TSP). TSP is NP-Complete and there is no even approximate algorithm with constant approximation ratio for it [6]. Thus the simple greedy walk strategy is adopted in GWSP, which is inspired by the *Anycast* mechanism. Moreover, the greedy walk strategy is also easy to implement in the distributed and asynchronous MANET environment. We also find that GWSP is load-balanced. The data source node and the caching nodes cooperate to propagate the data updates and hence amortize the overhead. The load balance of GWSP makes it suitable for the resource-constrained mobile terminals.

3 Experimental Evaluation

3.1 Experimental Methodology and Configurations

We have conducted simulations to evaluate the performance of GWSP. In the experiments, we first compare *greedy walk* with the optimal solution. We adopt a brutal force search strategy to obtain the optimal solution. The network we consider should contain a suitable number of nodes, so that computation of the optimal solution does not become exceedingly cumbersome. Then we finely tune the number of caching nodes and the cache query rate, which have great impact on GWSP. We study the cost-effectiveness of GWSP based on extensive performance comparison with the *Pull with Dynamic TTR* (named as DynTTR in short) algorithm [11, 12, 13, 20]. The following performance metrics are used in the evaluation:

- Traffic overhead: number of hops counted for consistency maintenance message propagation;
- Query latency: latency imposed by consistency maintenance. We obtain the query latency based on the number of hops the message needs to be relayed in the MANET;

Detailed experimental configurations are listed in Table 1.

Table 1. Experiment configurations.

Network area	200×200 m^2
Size of network	80 m
Transmission range	15 m
Mobility model	Random way point [23]
Average speed	0.5 m/s
Maximum portion of crashed nodes	10%
Pattern of date updates and queries	*Poisson process*
Initial value of TTR (δ)	10 s
τ_u	50%
τ_q	50%
τ_{ACK}	2s

The default configurations of the number of caching nodes and average query interval are as follows:

- number of caching nodes: 20 ;
- average query interval: 10 s ;

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

3.2 Evaluating the Greedy Walk Strategy

In this experiment, we compare greedy walk with the optimal solution. From the evaluation results (Fig 3), we find that, even compared with the optimal solution, the greedy walk strategy is quite cost-effective, especially when there are a suitable number of caching nodes. As the number of caching nodes increase, the performance of greedy walk gradually degrades. Given the computational complexity of finding the optimal solution, greedy walk is quite cost-effective.

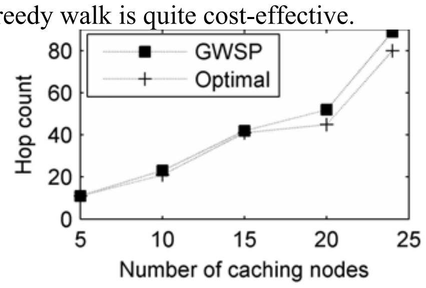
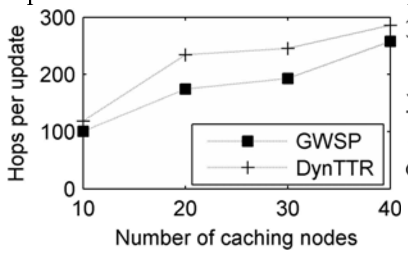


Fig. 3. Cost-effectiveness of greedy walk.

3.3 Effects of Tuning the Number of Caching Nodes

In this experiment, we increase the number of caching node from 10 to 20, 30 and 40. We find that GWSP is comparatively more cost-effective when there are a suitable number of caching nodes. When there are 20 and 30 caching nodes, the traffic overhead saved by GWSP is 26% and 21% respectively (Fig. 4). Here, the percentage of traffic overhead saved refers to the ratio of the saved traffic overhead to that imposed by DynTTR. However, when there are 10 and 40 caching nodes, the traffic overhead saved decreases to 15% and 9% respectively (Fig. 4). It is mainly because, when there are only a few caching nodes, little traffic overhead can be saved by selective push in GWSP. On the other hand, when there are many caching nodes, the PUSH & ACK process in GWSP also imposes great traffic overhead.

We also find that the query latency imposed by GWSP is relatively stable, while the latency imposed by DynTTR increases as the number of caching nodes increases (Fig. 5). The query latency saved in GWSP can go up to 85% when there are 30 caching nodes. The selective push mechanism based on the cache status accounts for the better performance of GWSP in terms of query latency.



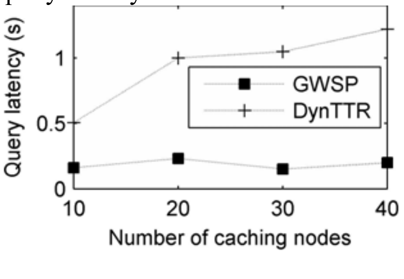


Fig. 4. Traffic overhead. **Fig. 5**. Query latency

3.4 Effects of Tuning the Cache Query Rate

In this experiment, we tune the cache query rate. We find that, when the cache query rate decreases, the traffic overhead imposed by GWSP gradually decreases, while the traffic overhead imposed by DynTTR decreases much more quickly (Fig. 6). The traffic overhead saved by GWSP decreases from 41% (query interval = 5s) to 19% (query interval = 20s). The query latency imposed by DynTTR decreases similarly. In GWSP, the query latency even increases when there are less frequent queries (query interval = 15), as shown in Fig. 7. The query latency saved in GWSP decreases from 72% (query interval = 5s) to 19% (query interval = 20s). It is mainly because less and less caching nodes are selected in the push set when faced with less and less frequent queries. Thus, the GWSP algorithm gradually approaches the Pull Each Read [14] algorithm, which suffers from round-traffic query latency.

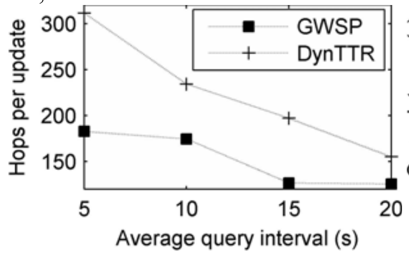


Fig. 6. Traffic overhead.

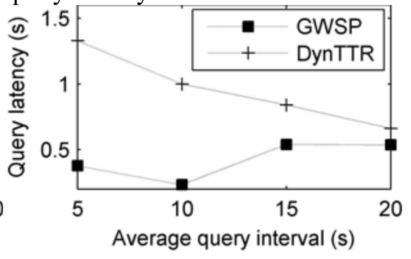


Fig. 7. Query latency

4 Conclusion and Future Work

In this paper, we study how to cost-effectively maintain cache consistency in cooperative caching in MANETs. We focus on stateful cache consistency maintenance algorithms since they can significantly reduce the consistency maintenance cost utilizing the cache status information. We propose the stateful *Greedy walk-based Selective Push* algorithm. In GWSP, the data source node maintains the TTR value and the cache query rate associated with each cache copy. Upon each source data update, the data source node efficiently decides the caching nodes which are in great need of the data updates. GWSP employs the greedy walk strategy to propagate the data updates among the selected caching nodes. Extensive experiments are conducted to evaluate the performance of GWSP. The evaluation results show that the greedy walk strategy can cost-effectively propagate the data updates in a MANET environment. The GWSP algorithm can save up to 41% traffic overhead and 85% query latency for cache consistency maintenance, compared with the *Pull with Dynamic TTR* algorithm. In our future work, we plan to study how to support different consistency models, besides Delta Consistency, by stateful consistency maintenance algorithms in cooperative caching in MANETs.

Acknowledgements

This work is supported in part by the National Natural Science Foundation of China under Grant No. 60673123, the National High-Tech Research and Development Program of China under Grant No. 2006AA01Z231, the Hong Kong RGC CERG grant PolyU 5105/05E and Hong Kong PolyU ICRG grant G-YF61.

References

1. M. Corson, J. Macker and G. Cirincione, Internet-based Mobile Ad Hoc Networksing, in IEEE Internet Computing, pp.63-70, July-August, 1999.
2. 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.
3. L. Yin and G. Cao, Supporting Cooperative Caching in Ad Hoc Networks, IEEE Transactions on Mobile Computing, Vol. 5, No. 1, 2006.
4. 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.
5. F. Sailhan and V. Issarny, Cooperative Caching in Ad hoc Networks, IEEE International Conference on Mobile Data Management (MDM), 2003.
6. T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, Chapter 35, Approximation Algorithms, Introduction to Algorithms, MIT Press, 2002.
7. Z. Wang, S.K. Das, H. Che and M. Kumar, A Scalable Asynchronous Cache Consistency Scheme (SACCS) for Mobile Environments, IEEE Tran. on Parallel and Distributed Systems, Vol. 15, No. 11, November 2004.
8. K. Tan, J. Cai and B. C. Ooi, An Evaluation of Cache Invalidation Strategies in Wireless Environments, IEEE Tran. on Parallel and Distributed Systems, Vol. 12, No. 8, Aug. 2001.
9. 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-21), Phoenix, AZ, April 2001.
10. L. Ferrigno, S. Marano, V. Paciello, A. Pietrosanto, Balancing Computational and Transmission Power Consumption in Wireless Image Sensor Networks", IEEE Symposium on Virtual Environments, Human-Computer Interfaces and Measurement Systems(VECIMS), 2005.
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-21), 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. 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 section, ACM Press, New York, USA, 2006.

14. J. Howard, M. Kazar, et al, Scale and Performance in a Distributed File System, ACM Trans. on Computer Systems, Vol. 6, No. 1, 1988.
15. D. Barbara and T. Imielinksi, Sleeper and Workaholics: Caching Strategy in Mobile Environments, Proc. ACM SIGMOD Conf. Management of Data, pp. 1-12, 1994.
16. G. Cao, A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments, Proc. ACM Int'l Conf. Computing and Networking (Mobicom), pp. 200-209, Aug. 2001.
17. A. Kahol, S. Khurana, S.K.S. Gupta, and P.K. Srimani, A Strategy to Manage Cache Consistency in a Distributed Mobile Wireless Environment, IEEE Trans. Parallel and Distributed Systems, vol. 12, no. 7, pp. 686-700, July 2001.
18. B. Yang, A.R. Hurson and Y. Jiao, On the Content Predictability of Cooperative Image Caching in Ad hoc Networks, in Proc. Of the 7th Intl. Conf. on Mobile Data Management (MDM), 2006.
19. B. Liu, W. Lee and D.L. Lee, Distributed Caching of Multi-dimensional Data in Mobile Environments, in Proc. Of the 6th Intl. Conf. on Mobile Data Management (MDM), 2005.
20. Y. Huang, J. Cao, Z. Wang, B. Jin and Y. Feng, Achieving Flexible Cache Consistency for Pervasive Internet Access, in proc. of the 5th Annual IEEE Intl. Conf. on Pervasive Computing and Communications (PerCom), pp. 239-250, New York City, U.S., 2007.
21. J. Cao, Y. Zhang, L. Xie and G. Cao, Data Consistency for Cooperative Caching in Mobile Environments, IEEE Computer, pp.60-67, April 2007.
22. Y.T.Hou, J.Pan, B.Li and S.S. Panwar, On Expiration-based Hierarchical Caching Systems, IEEE J. on Selected Areas in Comm., Vol.22, No.1, Jan 2004.
23. 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.