Cache Consistency in Mobile Ad-hoc Networks with SSUM

Mahesh Akuthota[1], **C.Srinivas**[2]

[1]M.Tech (Software Engineering), Kakatiya Institute of Technology & Science, Warangal, India

[2]Associate Professor of CSE Department, Kakatiya Institute of Technology & Science, Warangal, India

Abstract: ***This paper proposes the Smart Server Update Mechanism for Maintaining Cache Consistency in Mobile Environments, the server autonomously sends data updates to the Cache Nodes(CN), meaning that it has to keep track of which CNs cache which data items. This can be done using a simple table in which an entry consists of the id of a data item (or query) and the address of the CN that caches the data. A node that desires a data item sends its request to its nearest Query Directory (QD). If this QD finds the query i≠n its cache, it forwards the request to the CN caching the item, which, in turn, sends the item to the requesting node (RN). Otherwise, it forwards it to its nearest QD, which has not received the request yet. If the request traverses all QDs without being found, a miss occurs and it gets forwarded to the server which sends the data item to the RN. In the latter case, after the RN receives the confirmation from the last traversed QD that it has cached the query, it becomes a CN for this data item and associates the address of this QD with the item and then sends a Server Cache Update Packet (SCUP) to the server, which, in turn, adds the CN's address to the data item in its memory. This setup allows the server to send updates to the CNs directly whenever the data items are updated***

Keywords: Data caching, cache consistency, invalidation, server-based approach, MANET.

1. Introduction

In a mobile ad hoc network (MANET), data caching is essential as it reduces contention in the network, increases the probability of nodes getting desired data, and improves system performance. The major issue that faces cache management is the maintenance of data consistency between the client cache and the server. In a MANET, all messages sent between the server and the cache are subject to network delays, thus, impeding consistency by download delays that are considerably noticeable and more severe in wireless mobile devices.

All cache consistency algorithms are developed with the same goal in mind: to increase the probability of serving data items from the cache that are identical to those on the server. A large number of such algorithms have been proposed in the literature, and they fall into three groups: server invalidation, client polling, and time to live (TTL). With server invalidation, the server sends a report upon each update to the client. Two examples are the Piggyback server invalidation and the Invalidation report mechanisms. In client polling, like the Piggyback cache validation of a validation request is initiated according to a schedule. If the copy is up to date, the server informs the client that the data have not been modified; else the update is sent to the client. Finally, with TTL algorithms, a server-assigned TTL value (e.g., T) is stored alongside each data item d in the cache. The data d are considered valid until T time units pass since the cache update. Usually, the first request for d submitted by a client after the TTL expiration will be treated as a miss and will cause a trip to the server to fetch a fresh copy of d. Many algorithms were proposed to determine TTL values, including the fixed TTL approach, adaptive TTL, and Squid's LM-factor. TTL-based consistency algorithms are popular due to their simplicity, sufficiently good performance, and flexibility to assign TTL values for individual data items. However, TTL-based algorithms, like client polling algorithms, are weakly consistent, in contrast to server invalidation schemes that are generally strongly consistent. According to, with strong consistency algorithms, users are served strictly fresh data items, while with weak algorithms, there is a possibility that users may get inconsistent (stale) copies of the data.

Table 1: Packets used in COACS after Integrating SSUM into it

Packet	Function	Description
HELLO	New node's arrival	Broadcasted by a newly arriving node
CSP	Node score request	Sent when a new QD is needed
CIP	COACS Info	Broadcasts the QD list to all nodes when the list changes
DRP	Data Request	Submitted by an RN to request data and forwarded by a QD to the CN after a hit, or to the server after a miss
DREP	Data Reply	Submitted by a CN or the server to the RN
QCRP	Query Caching Request	Sent by a CN to a QD to cache one or more queries
EDP	Entry Deletion	Deletes a QD entry for a particular CN
CACK	Cache Add Acknowledge	Sent from a QD to a CN to Acknowledge caching
QDAP	QD Assignment Packet	Invitation to a node to become a QD
RUEP	Remove Update Entry form the server	Sent by a QD to the server to remove the address of a CN from one or more data items
SCUP	Server Cache Update Packet	Sent by the CN to the server asking it to set its address as the CN caching a certain data item
CICP	Update the CN cache	Sent by a CN to a QD or to the server to update its cache
CIRP	Update the CN cache	Reply to the CICP sent by a QD or the server to the CN

This work describes a server-based scheme implemented on top of the COACS caching architecture we proposed in In COACS, elected query directory (QD) nodes cache submitted queries and use them as indexes consistency strategy, the system described in this paper fills that void and adds several improvements: 1) enabling the server to be aware of the cache distribution in the MANET, 2) making the cached data items consistent with their version at the server, and 3) adapting the cache update process to the data update rate at the server relative to the request rate by the clients. With these changes, the overall design provides a complete caching system in which the server sends to the client's selective updates that adapt to their needs and reduces the average query response time.

2. Smart Server Update Mechanism (SSUM)

SSUM is a server-based approach that avoids many issues associated with push-based cache consistency approaches. Specifically, traditional server-based schemes are not usually aware of what data items are currently cached, as they might have been replaced or deleted from the network due to node disconnections. Also, if the server data update rate is high relative to the nodes request rate, unnecessary network traffic would be generated, which could increase packet dropout rate and cause longer delays in answering node queries. SSUM reduces wireless traffic by tuning the cache update rate to the request rate for the cached data.

2.1 Basic Operations

Before detailing the operations of SSUM, we list in Table 1 the messages used in the COACS architecture as we refer to them in the paper. Given that no consistency mechanism was implemented in COACS, it was necessary to introduce four additional messages. In SSUM, the server autonomously sends data updates to the CNs, meaning that it has to keep track of which CNs cache which data items. This can be done using a simple table in which an entry consists of the id of a data item (or query) and the address of the CN that caches the data. A node that desires a data item sends its request to its nearest QD. If this QD finds the query in its cache, it forwards the request to the CN caching the Item, which, in turn, sends the item to the requesting node (RN). Otherwise, it forwards it to its nearest QD, which has not received the request yet. If the request traverses all QDs without being found, a miss occurs and it gets forwarded to the server which sends the data item to the RN. In the latter case, after the RN receives the confirmation from the last traversed QD that it has cached the query, it becomes a CN for this data item and associates the address of this QD with the item and then sends a Server Cache Update Packet (SCUP) to the server, which, in turn, adds the CN's address to the data item in its memory. This setup allows the server to send updates to the CNs directly whenever the data items are updated. Fig. 1 illustrates few data request and update scenarios that are described below. In the figure, the requesting nodes (RNs) submit queries to their nearest QDs, as shown in the cases of RN1, RN2, and RN3. The query of RN1 was found in QD1, and so the latter forwarded the request to CN1, which returned the data directly to the RN. However, the query of RN2 was not found in any of the QDs, which prompted the last searched (QD1) to forward the request to the server, which, in turn, replied to RN2 that became a CN for this data afterward. The figure also shows data updates (key data pairs) sent from the server to some of the CNs.

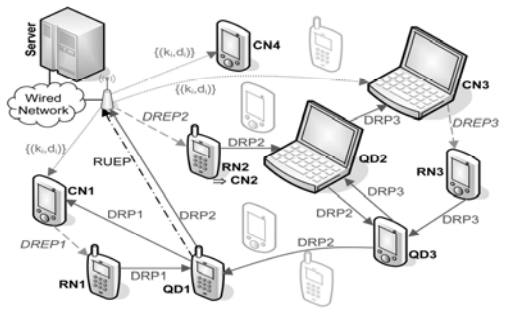
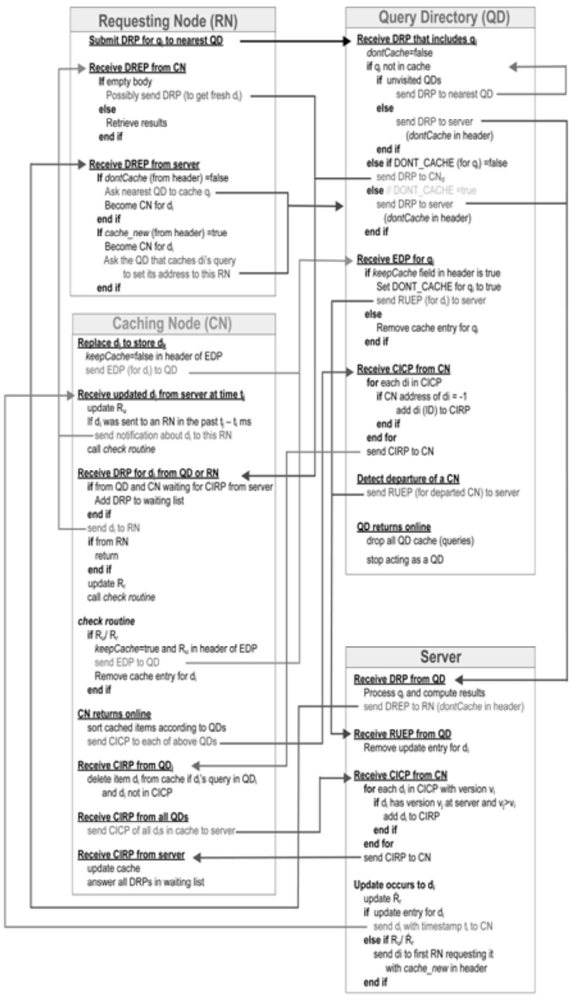


Figure 1: Scenarios for requesting and getting data in the COACS architecture.

2.2 Dealing with Query Replacements and Node Disconnections

A potential issue concerns the server sending the CN updates for data that have been deleted (replaced), or sending the data out to a CN that has gone offline. To avoid this and reduce network traffic, cache updates can be stopped by sending the server Remove Update Entry Packets (RUEPs). This could occur in several scenarios. For example, if a CN leaves the network, the QD, which first tries to forward it a request and fails, will set the addresses of all queries whose items are cached by this unreachable CN in its cache to 1, and sends an RUEP to the server containing the IDs of these queries. The server, in turn, changes the address of that CN in its cache to 1 and stops sending updates for these items. Later, if another node A requests and then caches one of these items, the server, upon receiving an SCUP from A, will associate A with this data item. Also, if a CN runs out of space when trying to cache a new item in, it applies a replacement mechanism to replace id within and instructs the QD that caches the query associated with id to delete its entry. This causes the QD to send an RUEP to the server to stop sending updates for id in the future. If a caching node CNd returns to the MANET after disconnecting, it sends a Cache Invalidation Check Packet (CICP) to each QD that caches queries associated with items held by this CN. A QD that receives a CICP checks for each item to see if it is cached by another node and then sends a Cache Invalidation Reply Packet (CIRP) to CNd containing all items not cached by other nodes. CNd then deletes from its cache those items who's IDs are not in the CIRP but were in the CICP. After receiving a CIRP from all QDs to which it sent a CICP and deleting nonessential data items from its cache, CNd sends a CICP containing the IDs of all queries with data remaining in its cache to the server along with their versions. In the meanwhile, if CNd receives a request from a QD for an item in its cache, it adds the request to a waiting list. The server then creates a CIRP and includes in it fresh copies of the outdated items and sends it to CNd, which, in turn, updates its cache and answers all pending requests. Finally, QD disconnections and reconnections do not alter the cache of the CNs, and hence, the pointers that the server holds to the CNs remain valid.



```
Requesting Node (RN)
Submit DRP for q_i to nearest QD

Receive DREP from CN
  If empty body
    Possibly send DRP (to get fresh d_i)
  else
    Retrieve results
  end if

Receive DREP from server
  If dontCache (from header) =false
    Ask nearest QD to cache q_i
    Become CN for d_i
  end if
  If cache_new (from header) =true
    Become CN for d_i
    Ask the QD that caches di's query
        to set its address to this RN
  end if
```

```
Caching Node (CN)
Replace d_i to store d_j
  keepCache=false in header of EDP
  send EDP (for d_i) to QD

Receive updated d_i from server at time t_i
  update R_u
  If d_i was sent to an RN in the past t_i – t_i ms
    send notification about d_i to this RN
  call check routine

Receive DRP for d_i from QD or RN
  if from QD and CN waiting for CIRP from server
    Add DRP to waiting list
  end if
  send d_i to RN
  if from RN
    return
  end if
  update R_r
  call check routine

check routine
  if R_u/ R_r
    keepCache=true and R_u in header of EDP
    send EDP to QD
    Remove cache entry for d_i
  end if

CN returns online
  sort cached items according to QDs
  send CICP to each of above QDs

Receive CIRP from QD_j
  delete item d_i from cache if d_i's query in QD_j
      and d_i not in CICP

Receive CIRP from all QDs
  send CICP of all d_is in cache to server

Receive CIRP from server
  update cache
  answer all DRPs in waiting list
```

```
Query Directory (QD)
Receive DRP that includes q_i
  dontCache=false
  if q_i not in cache
    if unvisited QDs
        send DRP to nearest QD
    else
        send DRP to server
          (dontCache in header)
    end if
  else if DONT_CACHE (for q_i) =false
    send DRP to CN_q
  else // DONT_CACHE =true
    send DRP to server
      (dontCache in header)
  end if

Receive EDP for q_i
  if keepCache field in header is true
    Set DONT_CACHE for q_i to true
    send RUEP (for d_i) to server
  else
    Remove cache entry for q_i
  end if

Receive CICP from CN
  for each di in CICP
    if CN address of di = -1
        add di (ID) to CIRP
    end if
  end for
  send CIRP to CN

Detect departure of a CN
  send RUEP (for departed CN) to server

QD returns online
  drop all QD cache (queries)
  stop acting as a QD
```

```
Server
Receive DRP from QD
  Process q_i and compute results
  send DREP to RN (dontCache in header)

Receive RUEP from QD
  Remove update entry for d_i

Receive CICP from CN
  for each d_i in CICP with version v_i
    if d_i has version v_j at server and v_j>v_i
      add d_i to CIRP
    end if
  end for
  send CIRP to CN

Update occurs to d_i
  update R̂_r
  if update entry for d_i
    send d_i with timestamp t_i to CN
  else if R_u/ R̂_r
    send di to first RN requesting it
      with cache_new in header
  end if
```

Figure 2: Detailed operations of the entities of SSUM using pseudo code.

3. Analysis

We evaluate our scheme in terms of bandwidth and query response time gains. These are the differences between the corresponding measures when no cache updating is in place and when SSUM is employed. Requests for data in the ad hoc network and data updates at the server are assumed to be random processes and may be represented by exponential random variables and we use λ_R to denote the rate of requests and λ_U the rate of updates. The probability density functions of requests and updates are thus given:

$P_R(t)=\lambda_R e^{-\lambda_R t}, P_U(t)= \lambda_U e^{-\lambda_U t}$

Both the bandwidth gain Gb and response time gain Gt are influenced by the number of data requests issued by requesting nodes relative to the number of data updates that occur at the server. In the remainder of the paper, we refer to no cache updating as NCU.

We now consider two cases: $\lambda_U /\lambda_R < 1$ (C1) and $\lambda_U/\lambda_R \geq 1$

(C2). C2, in turn, comprises two scenarios: after $\lambda_U/\lambda_R \geq \tau$ and then while $\lambda_U /\lambda_R \geq\gamma$ (C2S2), where updates are suspended by the server, and the remaining scenario (C2S1), where updates are sent by the server (illustrated in Fig. 3).

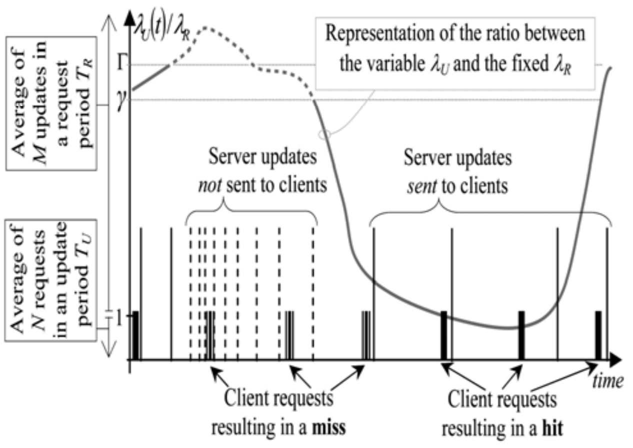


Figure 3: Illustration of the behavior of SSUM as $\lambda_U = \lambda_R$ changes (λ_R is shown fixed).

4. Performance Evaluation

We used the ns2 software to implement the SSUM system, and to also simulate the Updated Invalidation Report (UIR) mechanism [10] in addition to a version of it that is implemented on top of COACS so that a presumably fairer comparison with SSUM is done. To start with, we list the default values of the main simulation parameters in Table 2

Table 2: Summary of Simulation Parameters' Values

Simulation Parameter	Default Value
Simulation time	2000 sec
Network size	750×750 m^2
Wireless bandwidth	2 Mb/s
Node transmission range (r_0)	100 m
Number of nodes	100
Node mobility model	Random Way Point
Node speed (v)	2 (m/s)
Node pause time	30 sec
Node disconnection rate	0.5/min
Total number of data items	10,000
Size of data item	1 – 10 Kb
Number of data items updated/sec	20
Delay at the data source	40 ms
Node request period	20 sec
Node request pattern	Zipf distribution (θ=0.5)
Node caching capacity	200 Kb
Update thresholds Γ and γ	1.25 and 0.75

4.1 Network and Cache Simulation Parameters

A single database server is connected to the wireless network through a fixed access point, while the mobile nodes are randomly distributed. The client cache size was fixed to 200 Kb, meaning that a CN can cache between 20 and 200 items, while the QD cache size was set to 300 Kb, and therefore, a QD can cache about 600 queries. We used the least recently used (LRU) cache replacement policy when the cache is full and a data item needs to be cached. Each scenario started with electing one QD, but more were elected when space was needed. Each scenario lasted for 2,000 seconds and repeated 10 times with the seed of the simulation set to a new value each time, and the final result was taken as the average of the 10 runs.

The SSUM system was implemented as a new C++ agent in ns2 that gets attached to the node class in the tcl code at simulation runtime. This implementation includes a cache class that defines and sets the needed data items as well as the operations of the caching methods that were described. Also, the routing protocols in ns2 were modified to process the SSUM packets and to implement the functions of the MDPF algorithm used for traversing the QD system. Other changes to the ns2 C++ code included modifying the packet header information which is used to control the cache update process. After implementing the changes in the C++ code, tcl scripts were written to run the various described scenarios.

4.2 The Query Model Parameters

The client query model was chosen such that each node in the network generates a new request every Tq seconds. When the simulation starts, each node generates a new request, and after Tq seconds, it checks if it has not received a response for the request it generated in which case, it discards it and generates a new request. We chose a default value for Tq equal to 20 seconds, but in order to examine the effect of the request rate on the system performance, we simulated several scenarios with various request rates. The process of generating a new request followed a Zipf-like access pattern, which has been used frequently to model non-uniform distributions. In Zipf law, an item ranked where ranges between 0 (uniform distribution) and 1(strict Zipf distribution). The default value of the zipf parameter was set to 0.5.Every second; the server updates a number of randomly chosen data items, equal to a default value of 20. The default values of and was set to 1.25 and 0.75, respectively, while the default number of node disconnections is 1 every two minutes with a period of 10 seconds, after which the node returns to the network. However, in the experiments that we present below, the above parameters' values are varied to study their effects on performance.

4.3 Data and Overhead Traffic

In this set of experiments, we separated the produced traffic into data traffic, which includes the requests plus forwarding and reply packets, and overhead traffic generated by CSP, CIP, SCUP, CICP, CIRP, and update packets. The graphs in Fig.4 show that the overhead traffic in SSUM makes up between 2 and 25 percent of the total traffic when the request rate is varied, and between 15 and 25 percent when the data update rate is changed. In all, it is shown that SSUM does not generally produce substantial overhead traffic.

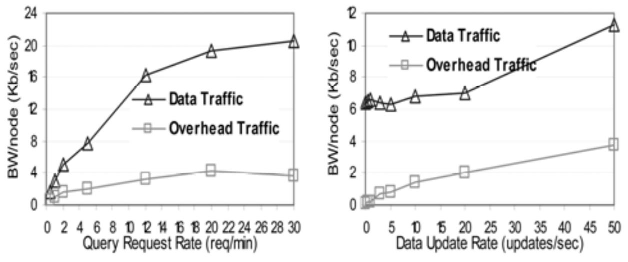


Figure 4: Data and overhead traffic of SSUM versus the request and update rates.

4.4 System Considerations

We discuss in this section some considerations for the implementation of SSUM that we derive from the system design and the simulation results. The major limitation which we point out is that an implementation of SSUM in a MANET where all nodes have short battery lifetime will not be efficient since this will result in frequent disconnections, and could lead to situations where no more nodes are qualified for becoming QDs. Hence, the requirement is that at any time, a portion of the nodes in the network must have sufficient energy to take on QD roles (in our simulations, the average number of QD nodes considered was 7 out of 100 nodes). This requirement though should be satisfied in most current MANETs since major technological improvements have been made to mobile device batteries, including those of cell phones. As an example, a survey of some recent cell phone models, including Nokia N86, Nokia N97, Sony Ericsson XPERIA X2, and Samsung I8000 revealed an average battery capacity of 430 hours in standby mode and 9 hours in talk time. Another issue is the network size. Similar to most server-based consistency schemes, SSUM will not be efficient in very large networks with thousands of nodes, where routing paths from the server to distant nodes could be lengthy, and more nodes will disconnect from and reconnect to the network. In these conditions, maintaining information about data items at all caching nodes will be expensive for the server. One of the possible solutions for such networks is the implementation of clustering in the MANET and adding more Access Points. Clustering can reduce overhead at the server since cluster heads can be used to represent the CNs within their clusters and act as single points of contact with the server. Additional APs will serve to reduce contention at the connections of the MANET to the wired network.

5. Conclusion and Future Works

This paper, we presented a novel mechanism for maintaining cache consistency in a Mobile Ad hoc Network. Our approach was built on top of the COACS architecture for caching data items in MANETs and searching for them. We analyzed the performance of the system through a gain- loss mathematical model and then evaluated it while comparing it with the Updated Invalidation Report mechanism. The presented results illustrated the advantage of our proposed system in most scenarios SSUM did not implement cache replication: Only one copy of the version at the server is maintained in the MANET. By adding replication to the design, the query delay is expected to go down but the overhead cost will surely go up. Need a tight security scheme. It is important that the nodes trust each other. Also, since QDs are the central component of the system, they could be the target of malicious attacks. Node is to be considered a candidate QD must have a high enough trust value

References

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik,

"Broadcast Disks: Data Management for Asymmetric Communications Environments," Proc. ACM SIGMOD, pp. 199-210, May 1995.

[2] 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.

[3] H. Artail and K. Mershad, "MDPF: Minimum Distance Packet Forwarding for Search Applications in Mobile AdHoc Networks," IEEE Trans. Mobile Computing, vol. 8, no. 10, pp. 1412- 1426, Oct. 2009.

[4] O. Bahat and A. Makowski, "Measuring Consistency in TTL based Caches," Performance Evaluation, vol. 62, pp. 439-455, 2005.

[5] D. Barbara and T. Imielinski, "Sleepers and Workaholics: Caching Strategies for Mobile Environments," Proc. ACM SIGMOD, pp. 1- 12, May 1994.

[6] C. Bettstetter and J. Eberspacher, "Hop Distances in Homogeneous Ad Hoc Networks," IEEE Proc. 57th IEEE Semiann. Vehicular Technology Conf., vol. 4, pp. 2286-2290, Apr. 2003.

[7] N.A. Boudriga and M.S. Obaidat, "Fault and Intrusion Tolerance in Wireless Ad Hoc Networks," Proc. IEEE Wireless Comm. And Networking Conf. (WCNC), vol. 4, pp. 2281-2286, 2005.

[8] Elmagarmid, J. Jing, A. Helal, and C. Lee, "Scalable Cache Invalidation Algorithms for Mobile DataAccess," IEEE Trans. Knowledge and Data Eng., vol. 15, no. 6, pp. 1498-1511, Nov. 2003.

[9] H. Jin, J. Cao, and S. Feng, "A Selective Push Algorithm for Cooperative Cache Consistency Maintenance overMANETs," Proc. Third IFIP Int'l Conf. Embedded and Ubiquitous Computing, Dec. 2007.

[10] IEEE Standard 802.11, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification, IEEE, 1999.

[11] J. Jing, A. Elmagarmid, A. Helal, and R. Alonso, "Bit-Sequences: An Adaptive Cache Invalidation Method inMobile Client/Server Environments," Mobile Networks and Applications, vol. 15, no. 2, pp. 115-127, 1997.

[12] J. Jung, A.W. Berger, and H. Balakrishnan, "Modeling TTL-Based Internet Caches," Proc. IEEE INFOCOM, Mar. 2003.

[13] X. Kai and Y. Lu, "Maintain Cache Consistency in Mobile Database Using Dynamical Periodical BroadcastingStrategy," Proc. Second Int'l Conf. Machine Learning and Cybernetics, pp. 2389- 2393, 2003.

[14] B. Krishnamurthy and C.E. Wills, "Piggyback Server Invalidation for Proxy Cache Coherency," Proc. Seventh World Wide Web (WWW) Conf., Apr. 1998.

[15] B. Krishnamurthy and C.E. Wills, "Study of Piggyback Cache Validation for Proxy Caches in the World Wide Web," Proc. USENIX Symp. Internet Technologies and Systems, Dec. 1997.

[16] D. Li, P. Cao, and M. Dahlin, "WCIP: Web Cache Invalidation Protocol," IETF Internet Draft, http://tools.ietf.org/html/draftdanli- wrec-wcip-01, Mar. 2001.

[17] W. Li, E. Chan, Y. Wang, and D. Chen, "Cache Invalidation Strategies for Mobile Ad Hoc Networks," Proc. Int'l Conf. Parallel Processing, Sept. 2007.

[18] M.N. Lima, A.L. dos Santos, and G. Pujolle, "A Survey of Survivability in Mobile Ad Hoc Networks," IEEE Comm. Surveys and Tutorials, vol. 11, no. 1, pp. 66-77, First Quarter 2009.