TÜBİTAK

Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2016) 24: 3459 – 3470
© TÜBİTAK
doi:10.3906/elk-1406-178

Push-pull cache consistency mechanism for cooperative caching in mobile ad hoc environments

Lilly Sheeba SELVIN*, **Yogesh PALANICHAMY**
Department of Information Science and Technology, Anna University, Chennai, Tamil Nadu, India

Received: 25.06.2014 • **Accepted/Published Online:** 29.04.2015 • **Final Version:** 20.06.2016

Abstract: Maintaining data consistency between data cached by a mobile client and data residing in the data server is a major issue in a wireless mobile environment. In this paper, we propose a push-pull cache consistency (2P2C) mechanism, with the intention of increasing the rate of serving requestors with recent up-to-date information or data from the cache of the intermediate nodes that are exact replicas of the data present in the origin server. This is a hybrid scheme that does not rely entirely either on the server or on the mobile client for maintaining consistency. Here we employ a registration process. Based on the registration process, the proposed 2P2C mechanism follows both a server-based consistency scheme and a client-based consistency scheme. Besides this, both the server and the cache nodes can legitimately accept client registrations. Hence, this 2P2C method highly decreases unnecessary communication overhead incurred in the network and thereby considerably reduces access latency.

Key words: Caching, consistency, ad hoc network, cache management, communication overhead

1. Introduction

In general, mobile wireless networks are broadly classified into either infrastructure-based wireless networks or infrastructure-less ad hoc networks. Infrastructure-dependent networks rely heavily on mobile support stations or access points (APs) to forward messages that the mobile nodes send or receive. On the other hand, mobile ad hoc networks (MANETs) use APs only to get connected to outside distributed networks like the Internet. Hence, infrastructure-based networks are confined with single-hop wireless communication while ad hoc networks depend on multiple hops for communication.

In MANETs, mobile devices are spread over an area in which access to external data is achieved through one or more nodes directly connected to APs. These directly connected nodes then act as routers to other nodes to access the APs. When many mobile nodes access these APs then heavy traffic may result in the network, leading to increased delay in receiving responses to queries. In addition, mobile networks face challenges like less stable wireless links and increased bandwidth utilization to cope up with the dynamic topology. Besides this, the user's mobility and energy utilization also play major roles in deciding the access latency and network overhead. In such a mobile network, it is inevitable that all nodes must mutually cooperate among themselves to accomplish various tasks efficiently.

Caching in a mobile environment becomes an attractive solution to improve the overall network performance by mitigating heavy traffic occurring near the data server and thereby reducing access latency incurred by queries. Frequently accessed data items can be cached in mobile devices. Cooperative caching enables

*Correspondence: slilly_sheeba@yahoo.com the cached data to be mutually shared among multiple mobile users. Hence, cooperative caching eliminates targeting the data server alone for all possible requests made by the mobile nodes. Moreover, delay incurred by requests traversing multiple hops to reach the data source can be eliminated by fetching cached data from nearby mobile devices.

The dynamic nature of this type of network makes storage and retrieval of cached information a difficult process. Moreover the cached information must be identical to the same information present in the data server. Efficient and effective techniques are needed to cache and manage information with consistency among the caches in a highly dynamic environment. An efficient cache system must provide an effective solution considering all these issues.

In this paper we propose a push-pull cache consistency (2P2C) method that tries to overcome all these drawbacks by adapting a registration process. The data server can be updated either randomly or periodically. The proposed mechanism is meant for an urban environment where the origin server is assumed to be a sports server, shopping mall server, etc. Based on the outcome of this process either a client-initiated cache consistency scheme or a server-initiated cache consistency scheme is invoked. The scheme adopts a dual push mechanism where the server pushes the data item to the desired cache nodes and the cache nodes in turn push the data to the requesting mobile clients, both based on registration.

2. Related work

Many caching schemes have been proposed by researchers as a result of various research efforts carried out to mitigate the serious issues that challenge mobile environments in terms of information retrieval. Several works related to caching have promised to deliver higher data availability [1–3] and strong cache consistency [4–6].

In general, cache consistency schemes can be classified into two major categories, namely push-based and pull-based schemes. While push-based schemes rely on the server for cache updates, pull-based schemes are client-based and attain cache updates on requests to the server. Pull-based schemes can be further categorized into client validation algorithms and time to live (TTL) algorithms.

In the case of client validation, a client sends validation requests to the server under certain criteria. If the data are not an exact copy of the same that are present with the server, then the server responds with a valid data message or else not a valid data message. A valid message implies that the data were not modified and an invalid message means that the data have undergone some changes. It is very much similar to piggyback cache validation [5]. In [7], an invalidation report (IR)-based cache invalidation algorithm, which can significantly reduce the query latency and efficiently utilize the broadcast bandwidth, was proposed. The IR-based cache invalidation leads to two major drawbacks. First, there is a long query latency associated with this solution since a client cannot answer the query until the next IR interval. Second, when the server updates a hot data item, all clients have to query the server and get the data from the server separately, which wastes a large amount of bandwidth.

On the other hand, TTL algorithms make use of TTL values assigned by the server. Every cache node stores this TTL value along with each datum in its cache. Many TTL-based algorithms have been proposed. A fixed TTL algorithm was described in [8]. In [9], Cao et al. proposed an adaptive TTL-based cache consistency scheme. In [10], an adaptive algorithm was discussed for maintaining consistency within a client server-based mobile network. Tang et al. [11] described a TTL-based consistency scheme maintained in unstructured peer-to-peer networks. In this network, millions of nodes share data through searching and replication. In this approach each replica is assigned a TTL value. Replication improves sharing of data but in turn complicates consistency.

For push-based schemes, two perfect examples are the piggyback invalidation approach proposed by Krishnamurthy et al. [4] and the IR method dealt with by Yin et al. [6]. Here the server sends a report to respective cache nodes based on update activities occurring on individual data items. Hence, the data present in the cache node and the server are exact replicas and hence consistency is guaranteed. Server-based approaches generally employ IRs that are periodically broadcasted by the server. An IR entry list normally carries the IDs of the updated data items and the time stamps of the updated history. When a query is generated from the requesting node, the node waits for the periodic IR to decide whether to invalidate its cache (if connected) or not. If it is valid, then the query is transmitted. If the requested data item is invalid or modified, it usually waits for the periodic IR. Cao et al. [12] proposed an IR-based algorithm, mainly directed to reduce network traffic. Server-based approaches generally employ IRs that are periodically broadcasted by the server.

3. Proposed system

A mobile ad hoc network is a wireless communication network, where nodes that are not within the direct transmission range of each other will require other nodes to forward data. It can operate without existing infrastructure, supports mobile users, and falls under the general scope of multihop wireless networking. It is a self-configured network of mobile terminals connected by wireless links. Mobile terminals such as cell phones, portable gaming devices, personal digital assistants (PDAs), and tablets all have wireless networking capabilities. Such heterogeneous mobile devices come together to collectively form a mobile wireless network. By participating in the network, these terminals may reach the Internet when they are not in the range of Wi-Fi access points or cellular base stations, or communicate with each other when no networking infrastructure is available. These types of networks find their importance in applications like military, disaster rescue and recovery, etc. However, the application domain of our work is an urban environment where data servers are involved. The nodes in such networks in turn rely on the data server for accessing data of interest.

Many of the cache consistency schemes either adopt push-based or pull-based techniques. For instance, [13] and [4] use push-based server update mechanisms while [14] and [5] rely on pull-based cache-initiated consistency management schemes.

However, sticking to any single update policy may lead to negative impacts on the autonomous and dynamic nature of the MANET environment. Hence, cache consistency schemes should concentrate on adopting either of the policies, as and when needed, or as per the current requirements.

We discuss this issue with respect to two different cases: 1) The client does not want to receive frequent updates related to any data item and wishes to receive update data only after requests. 2) The client wishes to receive all recent updates of any particular data item without making frequent requests.

On considering Case 1, we can conclude from our previous studies that adopting a pull-based client initiated update policy may be an apt choice, while for Case 2, going for a push-based server update scheme might be apt. Hence, a good cache consistency mechanism should provide solutions considering these two valid scenarios. If not, clients will be forced to receive updates even if they do not intend to receive them.

Henceforth, in this paper, we move on to propose a 2P2C scheme that allows clients to choose or opt for their own mode of update mechanisms. Here we employ a registration process, which the client initiates in its request to the server. In the request, the client informs the server whether it is for a push-based server-initiated or pull-based client-initiated policy. If it wishes to go for a push-based policy it sets the registration flag on, indicating the server to update it frequently based on current updates on the data item of its interest. On seeing this, the server makes an entry in its table and responds with updates, as and when required, to the client. If the registration bit is not set, it indicates that the client is making a one-time request and it is not interested in receiving recent updates made on the data. The client here is ready to pull the data as and when it desires.

3.1. Server-initiated consistency policy

Here the server maintains a server update table (SUT) for all data items for which it was hit by various cache nodes with registration bit set on. Table 1 shows the format of the SUT for a single data item.

Table 1. Format of SUT.

Data item	D_i
Time of update	T_{ui}
Time of next update	T_{Nui}
Time of refresh	T_{Ri}
Cache node list	CNL_i

Table 2 depicts the cache node list (CNL) for data item D_i, CNL_i. Here X, Y, and Z are nodes that have cached the data item D_i. T_{Xi}, T_{Yi}, and T_{Zi} correspond to the time at which the requested data item was delivered to the desired nodes from the data server.

Table 2. Format of CNL.

Node that has cached D_i	X	Y	Z
Time of departure of D_i	T_{Xi}	T_{Yi}	T_{Zi}

The server releases an update packet after T_{Nui}. This T_{Nui} is calculated as in Eq. (1).

$$T_{Nui} = T_{ui} + T_{Ri} \tag{1}$$

Here the server responds to any client request with the desired data item along with time of departure and the time of the next update. For instance, for an item i, the server sends (D_i, T_{Xi}, T_{Nui}) to the requesting node X. At T_{Nui}, the server performs two jobs. First it changes the time of update with the time of the next update. This is expressed as in Eq. (2).

$$T_{ui} = T_{Nui} \tag{2}$$

Next it pushes the updated data item to the caches of the respective cache nodes. The time of push is given as:

$$T_{Nui} \leq T_{push} \leq T_{Nui} + T_{Ri} \tag{3}$$

One such full entry for data item D_i made in the SUT table on the server is clearly depicted in Table 3. CNL_i for data item D_i is shown in Figure 1.

Table 3. Entry in SUT.

D_i	T_{ui}	T_{Nui}	T_{Ri}	CNL_i

Table 4. Format of CUT.

Data item	D_i
Time of next update	T_{Nui}
Time of refresh	T_{Ri}
Request node list	RNL_i

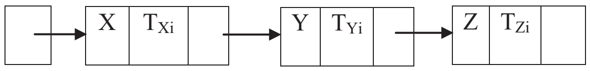
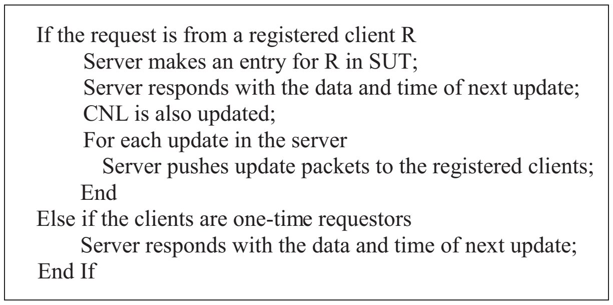


Figure 1. CNLi for D$_i$.

Here a minimum heap data structure as in [15] has been adopted by nodes to provide for sorting based on their time of departure. For instance:

$$T_{Xi} < T_{Yi} < T_{Zi} \tag{4}$$

The server side consistency scheme is depicted in Figure 2.



```
If the request is from a registered client R
    Server makes an entry for R in SUT;
    Server responds with the data and time of next update;
    CNL is also updated;
    For each update in the server
        Server pushes update packets to the registered clients;
    End
Else if the clients are one-time requestors
    Server responds with the data and time of next update;
End If
```

Figure 2. Pseudocode for server-initiated consistency policy.

3.2. Client-initiated consistency policy

As in [16], once the requested data item reaches the requesting cache node, for instance X, it caches the data item along with the next update time and time of departure. Every node maintains a cache node information table having details of all the other cache nodes in order to provide for cooperative caching similar to the index vector in [17]. This table has the parameters of the cache node ID and the list of items cached by the respective cache nodes.

Stale time T$_{Sxi}$ of item D$_i$ in a cache node X denotes the amount of time for which the data item remains static without any updates. It is computed as in Eq. (5).

$$T_{Sxi} = (T_{ui} + T_{Ri}) - T_{Xi} \tag{5}$$

Hence, using Eq. (1) in Eq. (4), we derive Eq. (6).

$$T_{Sxi} = T_{Nui} - T_{Xi} \tag{6}$$

T$_{Nui}$ is then used by the client to perform automatic deletion. On reaching the next update time of any data item, the requesting node removes the data item from its cache or else marks it for deletion, as otherwise the data might become inconsistent. The relationship among the various parameters involved in the deletion process is represented as:

$$T_{Nui} \leq T_{Di} \leq T_{Nui} + T_{Ri} \tag{7}$$

Here T$_{Di}$ is the time intended for deletion of any item.

Moreover, this client cache node can in turn serve the requests for the same data coming towards it from its neighbors, instead of always forwarding the request towards the server. In order to satisfy this requirement the cache nodes maintains a cache update table (CUT) for all data items for which it was hit by various requesting clients with registration bit set on. Table 4 shows the format of the CUT for a single data item.

Table 5 depicts the requesting node list for item i, RNL_i. Here R1, R2, and R3 are nodes that have requested data item D_i. T_{R1i}, T_{R2i}, and T_{R3i} correspond to the time at which R1, R2, and R3 have received their desired and requested data item, D_i, from the cache node, respectively.

Table 5. Format of RNL.

Node that has requested D_i from the cache node	R1	R2	R3

At soon as the cache node gets updated data packets from the server it immediately responds to all its registered clients with the update information of the desired data item. One such full entry for a data item D_i made in the CUT table on the server is clearly depicted in Table 6. RNL_i for data item D_i is shown in Figure 3.

Table 6. Entry in CUT.

D_i	T_{Nui}	T_{Ri}	RNL_i

The operations that are carried out to maintain consistency among the client nodes are shown in Figure 4.

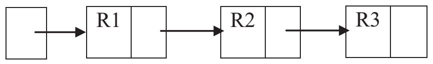
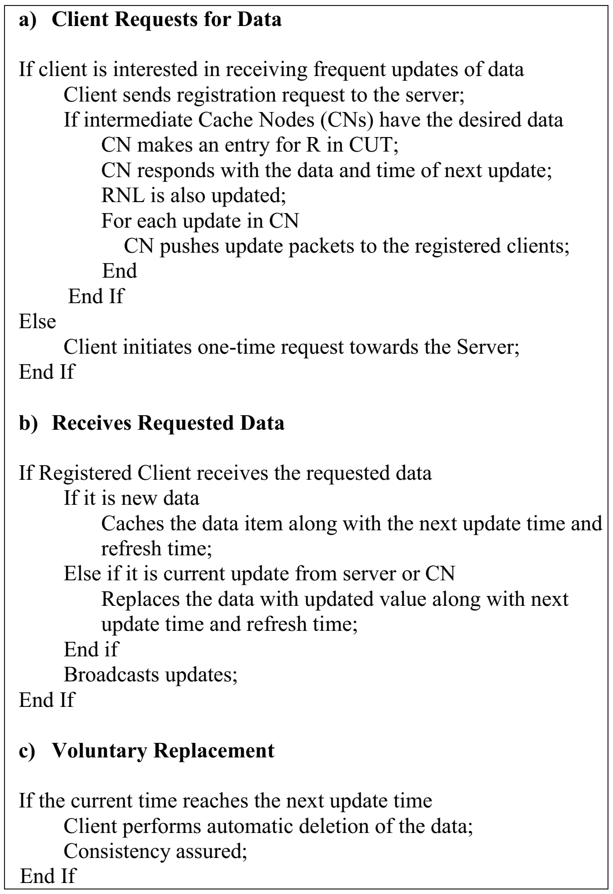


Figure 3. RQLi for D_i.



```
a)  Client Requests for Data

If client is interested in receiving frequent updates of data
      Client sends registration request to the server;
      If intermediate Cache Nodes (CNs) have the desired data
           CN makes an entry for R in CUT;
           CN responds with the data and time of next update;
           RNL is also updated;
           For each update in CN
              CN pushes update packets to the registered clients;
           End
       End If
Else
      Client initiates one-time request towards the Server;
End If

b)  Receives Requested Data

If Registered Client receives the requested data
      If it is new data
           Caches the data item along with the next update time and
           refresh time;
      Else if it is current update from server or CN
           Replaces the data with updated value along with next
           update time and refresh time;
      End if
      Broadcasts updates;
End If

c)  Voluntary Replacement

If the current time reaches the next update time
      Client performs automatic deletion of the data;
      Consistency assured;
End If
```

Figure 4. Pseudocode for client-initiated consistency policy.

3.3. Assumptions

- Nodes in MANET are resource-constrained devices. When a data reply is sent from the server back to the requesting client, all the intermediate and immediate forwarding nodes might be tempted to cache the data. Caching the same data within nearby neighbors might lead to wastage of cache space since available memory is limited in compact wireless devices when compared to devices like personal computers and other sophisticated machines. Hence, as in [18], only selected nodes are destined to cache data.
- The cache node selection can also take into account the weight-based algorithm [19] along with memory space availability. This avoids the possibility of data loss by choosing high mobility nodes with less space availability for caching.
- The server gets updated periodically and is aware of the occurrence of the next update.
- In order to maintain strong consistency between the cache node and server, the next update time and refresh time are used. Based on these time-related information, cached data have to be removed or updated. Hence, synchronization of timers is required among the server and other nodes.

3.4. Performance evaluation

Network Simulator Version 2 (NS2) [20] is used to simulate the proposed 2P2C cache scheme. NS2 is a network simulator that is widely employed in wired or wireless environments to simulate a layered environment. Our simulation settings and parameters are summarized in Table 7. Here, mobile nodes are made to move in a 1500 m $\times$ 1000 m area for a simulation time of 500 s. A random way point model is adopted in the simulation set up. For analysis, we consider traffic, delay, update rate, and cache size as performance metrics. These metrics highly aid in evaluating the effectiveness of the 2P2C scheme against nonregistered schemes. In short, the behavior of the system when the registration process is implemented can be well evaluated.

Table 7. Simulation parameters.

Simulation parameter	Value
Simulation time	500 s
Network area	1500×1000 m^2
Bandwidth	2 Mb/s
Transmission range	300 m
Number of nodes	0–300
Speed	0–30 m/s
Data items in the server	2000
Size of data item	10 KB
Cache size	100–600 KB
Updates/minute	100 updates/min
Delay	20 ms
Request rate	5 requests/min

3.4.1. Performance metrics

- Query request rate: The rate at which an item is requested in requests/minute from the server.
- Query delay: The delay between the query requests and the time at which the data arrive at the requesting node. It is measured in terms of seconds.

- Update delay: The delay between the update happening in the server and that being communicated to appropriate cache nodes for a particular data item in terms of seconds.
- Update rate: The rate at which any particular data item gets updated at the server. It is given in terms of updates/second.
- Traffic: This traffic includes requests for an item, forwarding of the requests, and replies for the requests. It is given in terms of kilobytes/second.
- Communication overhead: This is the overhead that arises due to transmissions of additional messages in order to maintain consistency between the cache clients and the server. It is again represented in terms of kilobytes/second.
- Hit ratio: This is used to define the success rate of requests generated within the network. Here we classify hits into three classes, namely:
- Local cache hit: This type of hit arises if the requested data are present in the cache of the requestor node itself.
- Cache node hit: If the requests are serviced by intermediate cache nodes, then it leads to a cache node hit.
- Server hit: In this case, all the previous attempts end up with a miss, and then the request is forwarded towards the server, which in turn serves the requestor with the desired data.

Hit ratio (HR) is in general given in terms of total requests serviced (TRS) and total requests generated (TRG) as in Eq. (8).

$$HR = \frac{TRS}{TRG} \tag{8}$$

It can also be given in terms of total unsuccessful requests (TUR) as in Eq. (9).

$$HR = \frac{TRG - TUR}{TRG} = \frac{1 - TUR}{TRG} \tag{9}$$

3.4.2. Simulation results

Here we compare our 2P2C scheme that involves registration with that of a system employing a nonregistered scheme, namely the smart server update mechanism (SSUM).

When the request rate for some particular data item is increased, then there is a considerable increase in query delay due to accumulation of more requests either in the server or in the cache nodes. This can be easily observed from Figure 5. However, when compared to the SSUM, query delay is reduced because both the server node and the cache node cooperate in the updating process.

In Figure 6, we find that the update delay decreases with increase in query request rate because the already updated nodes can serve the needs of the newly requesting nodes. Moreover, it can be noted that after some point of time the delay remains steady. The initial increase is attributed to the reception of updated data at the destined cache nodes.

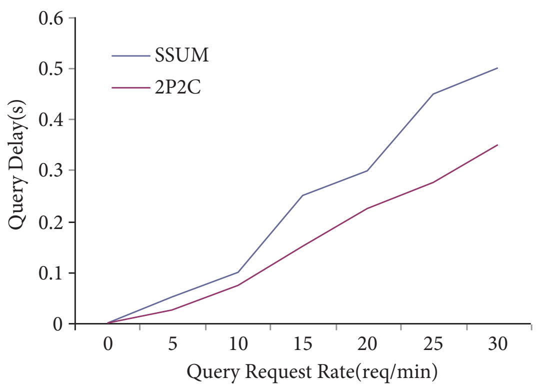


Figure 5. Query request rate vs. query delay.

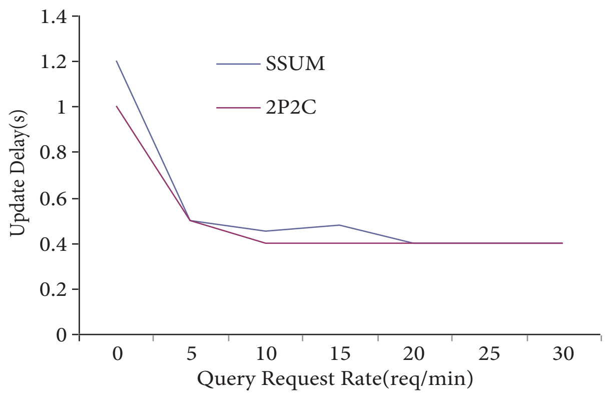


Figure 6. Query request rate vs. update delay.

From Figure 7, it can be easily ascertained that the update delay in the case of our 2P2C scheme is much less when compared to that of systems employing unregistered schemes. This is because both the server and the cache node take part in the registration process and serve their respective recipients with ease. Both of them push the desired data items to the respective registered clients, unlike the SSUM, which depends only on the server.

It can be understood from Figure 8 that as the update rate increases the traffic also increases, but much less so than with the SSUM. Traffic here involves forwarding of requests and sending the reply with the desired data back to the requesting node. Since here cache nodes can also serve the requesting nodes, unlike in the SSUM, the traffic is very much reduced. In addition, unregistered clients are restricted from sending update data periodically.

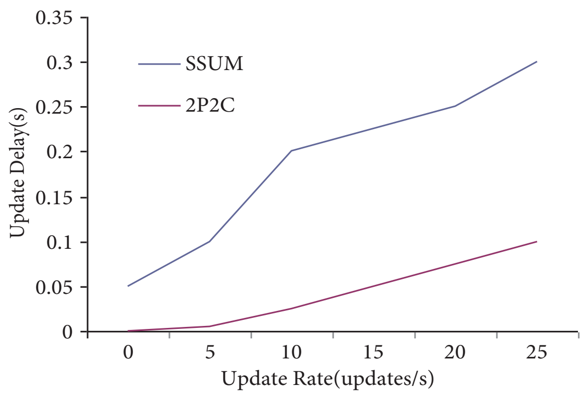


Figure 7. Update rate vs. update delay.

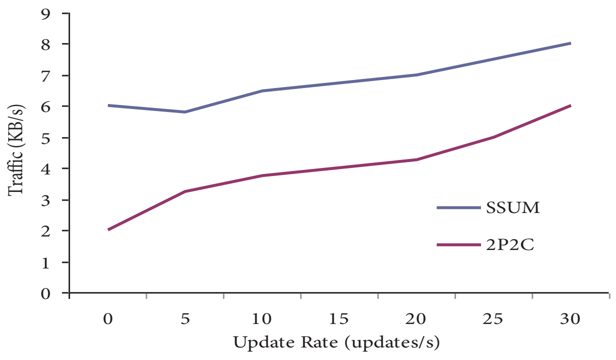


Figure 8. Update rate vs. traffic.

Communication overhead constitutes an exchange of additional messages. Since in 2P2C the registration process is packed along with the first request itself, additional messages are not involved as in the SSUM. Hence, after some point of time, as in Figure 9, communication overhead does not increase.

In Figure 10, velocity of mobile nodes is varied between 0 and 30 m/s. It can be inferred from the figure that as the speed is increased, query delay also increases. In the 2P2C scheme, cache nodes also serve the incoming requests from clients without interference of any special nodes like query directories present in the SSUM, resulting in reduced query delay.

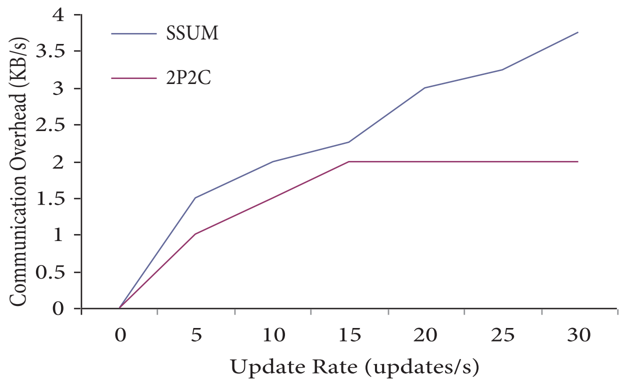


Figure 9. Update rate vs. communication overhead.

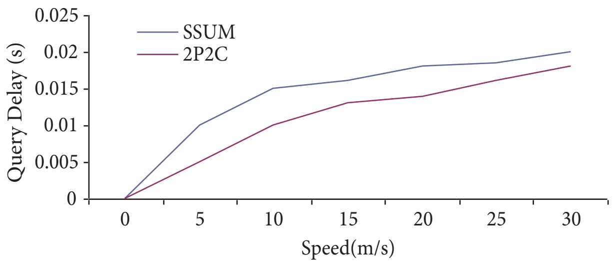


Figure 10. Speed vs. query delay.

It can be predicted from Figure 11 that as the velocity of nodes is increased, update delay also increases. However, in our proposed scheme, the delay is comparatively less as no special messages are employed for maintaining updates in nodes.

Figures 12 and 13 portray the effect of node density on query delay and update delay, respectively. Here it can be visualized that increasing the number of nodes in the network has considerable impact on both query and update delay. This effect is mainly due to cache nodes assisting the server and that update messages are periodically pushed to the cache nodes and from them to the requesting clients.

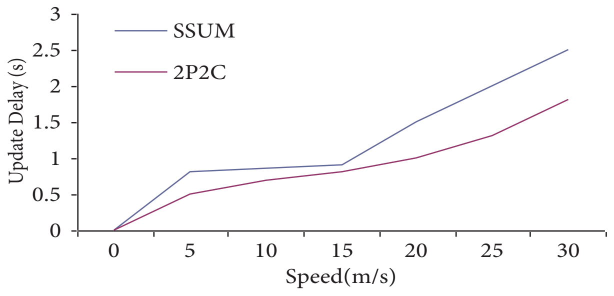


Figure 11. Speed vs. update delay.

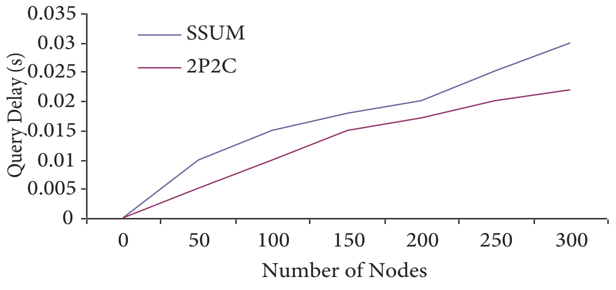


Figure 12. Number of nodes vs. query delay.

In Figure 14, the impact of various types of hits when the cache size is varied can be observed. As the cache size is varied between 100 KB and 600 KB, caching occurs in many nodes and these nodes can serve the incoming requests well in advance when compared to requests reaching the ultimate source, the server.

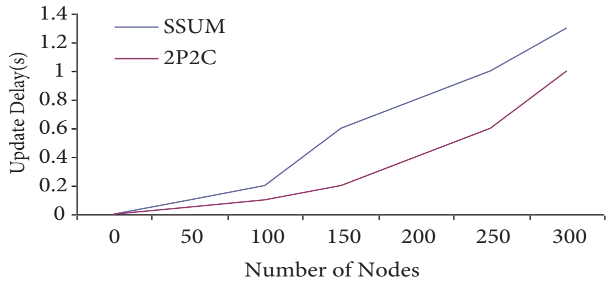


Figure 13. Numbers of nodes vs. update delay.

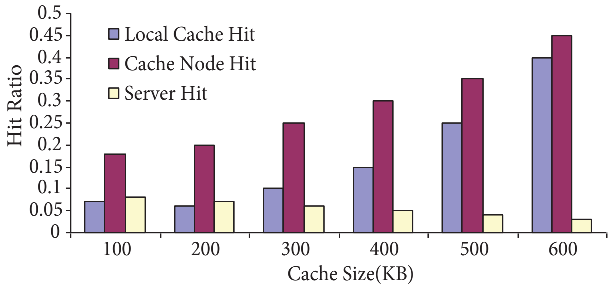


Figure 14. Cache size vs. hit ratio.

4. Conclusion

The proposed efficient cache consistency scheme devised in a cooperative cache wireless environment provides better data availability to mobile users but with an affinity to replicate identical information as maintained in the server at any point of time. The scheme is conscious enough to serve users with only current updates.

The 2P2C scheme achieves this by employing a registration process with an intention to accustom itself with either a server-initiated or client-initiated cache consistency method. This in turn reduces communication overhead in addition to mitigating heavy traffic that may arise near the server. Moreover, load balancing is also obtained since the server and the cache nodes extend their cooperation in the update process to bring about strong consistency.

References

[1] Yin L, Cao G. Supporting cooperative caching in ad hoc networks. IEEE T Mobile Comput 2006; 5: 77-89.

[2] Lim S, Lee W, Cao G, Das C. A novel caching scheme for improving internet based mobile ad hoc networks performance. Ad Hoc Netw 2006; 4: 225-239.

[3] Artail H, Safa H, Mershad K, Abou-Atme Z, Sulieman N. COACS: A cooperative and adaptive caching system for MANETs. IEEE T Mobile Comput 2008; 7: 961-977.

[4] Krishnamurthy B, Wills CE. Piggyback server invalidation for proxy cache coherency. In: Seventh International World-Wide Web Conference; 14–18 April 1998; Brisbane, Australia. pp. 185-193.

[5] Krishnamurthy B, Wills C. Study of piggyback cache validation for proxy caches in the World Wide Web. In: Usenix Symposium on Internet Technologies and Systems; 8–11 December 1997; California, USA. pp. 1-12.

[6] Yin L, Cao G, Cai Y. A generalized target driven cache replacement policy for mobile environments. In: International Symposium on Applications and the Internet; 27–31 January 2003; Orlando, FL, USA. pp. 14-21.

[7] Cao G. A scalable low-latency cache invalidation strategy for mobile environments. IEEE T Knowl Data En 2003; 15: 1251-1265.

[8] Jung J, Berger AW, Balakrishnan H. Modeling TTL-based internet caches. In: INFOCOM; 30 March–3 April 2003; San Francisco, CA, USA. pp. 417-426.

[9] Cao P, Liu C. Maintaining strong cache consistency in the world-wide web. IEEE T Comput 1998; 47: 445-457.

[10] Jing J, Elmagarmid A, Helal A, Alonso R Bit-sequences: an adaptive cache invalidation method in mobile client/server environments Mobile Netw Appl 1997; 2: 115-127.

[11] Tang X, Xu J, Lee WC. Analysis of TTL-based consistency in unstructured peer-to-peer networks. IEEE T Parall Distr 2008; 19: 1683-1694.

[12] Cao J, Zhang Y, Cao G, Li X. Data consistency for cooperative caching in mobile environments. Computer 2007; 40: 60-66.

[13] Mershad K, Artail H. SSUM: Smart server update mechanism for maintaining cache consistency in mobile environments. IEEE T Mobile Comput 2010; 9: 778-795.

[14] Fawaz K, Artail H. DCIM: Distributed cache invalidation method for maintaining cache consistency in wireless mobile networks. IEEE T Mobile Comput 2013; 12: 680-693.

[15] Sheeba LS, Yogesh P. An efficient HOT-B protocol for caching in mobile ad hoc networks. In: International Conference on Recent Trends in Information Technology; 19–21 April 2012; Chennai, India. pp. 326-331.

[16] Sheeba LS, Yogesh P. A time index based approach for cache sharing in mobile ad hoc networks. In: International Conference on Computer Science, Engineering and Applications; 15–17 July 2011; Chennai, India. pp. 1-8.

[17] Chiu GM, Young CR. Exploiting in-zone broadcasts for cache sharing in mobile ad hoc networks. IEEE T Mobile Comput 2009; 8: 384-397.

[18] Dimokas N, Katsaros D, Tassiulas L, Manolopoulos Y. High performance, low complexity cooperative caching for wireless sensor networks. Wirel Netw 2011; 17: 717-737.

[19] Chatterjee M, Sajal KD, Turgut D. WCA: A weighted clustering algorithm for mobile ad hoc networks. Cluster Comput 2002; 5: 193-204.

[20] Issariyakul T, Hossain E. Introduction to Network Simulator NS2. 2nd ed. New York, NY, USA: Springer, 2012.