DCIM: Distributed Cache Invalidation Method for Maintaining Cache Consistency in Wireless Mobile Networks

Kassem Fawaz, *Student Member*, *IEEE*, and Hassan Artail, *Senior Member*, *IEEE*

Abstract—This paper proposes distributed cache invalidation mechanism (DCIM), a client-based cache consistency scheme that is implemented on top of a previously proposed architecture for caching data items in mobile ad hoc networks (MANETs), namely COACS, where special nodes cache the queries and the addresses of the nodes that store the responses to these queries. We have also previously proposed a server-based consistency scheme, named SSUM, whereas in this paper, we introduce DCIM that is totally client-based. DCIM is a pull-based algorithm that implements adaptive time to live (TTL), piggybacking, and prefetching, and provides near strong consistency capabilities. Cached data items are assigned adaptive TTL values that correspond to their update rates at the data source, where items with expired TTL values are grouped in validation requests to the data source to refresh them, whereas unexpired ones but with high request rates are prefetched from the server. In this paper, DCIM is analyzed to assess the delay and bandwidth gains (or costs) when compared to polling every time and push-based schemes. DCIM was also implemented using ns2, and compared against client-based and server-based schemes to assess its performance experimentally. The consistency ratio, delay, and overhead traffic are reported versus several variables, where DCIM showed to be superior when compared to the other systems.

Index Terms—Cache consistency, data caching, client-based, invalidation, MANET, TTL

1 Introduction

Mobile devices are the building blocks of mobile ad hoc networks (MANETs). They are typically characterized by limited resources, high mobility, transient availability, and lack of direct access to the data source (server). In MANET environments, data caching is essential because it increases the ability of mobile devices to access desired data, and improves overall system performance [15], [22]. In a typical caching architecture, several mobile devices cache data that other devices frequently access or query. Data items are essentially an abstraction of application data that can be anything ranging from database records, webpages, ftp files, etc.

The major issue that faces client cache management concerns the maintenance of data consistency between the cache client and the data source [3]. All cache consistency algorithms seek to increase the probability of serving from the cache data items that are identical to those on the server. However, achieving strong consistency, where cached items are identical to those on the server, requires costly communications with the server to validate (renew) cached items, considering the resource limited mobile devices and the wireless environments they operate in. Consequently there exist different consistency levels describing the degree to which the cached data is up to date. These levels, other than strong consistency, are weak consistency, delta consistency [5], [6], probabilistic consistency [8], [11], and probabilistic delta consistency [13].

With weak consistency, client queries might get served with inconsistent (stale) data items, while in delta consistency, cached data items are stale for up to a period of time denoted as *delta*. In probabilistic consistency, a data item is consistent with the source with a certain probability denoted as p. Finally, in probabilistic delta consistency, a certain cached item is at most *delta* units of time stale with a probability not less than p.

The cache consistency mechanisms in the literature can be grouped into three main categories: push based, pull based, and hybrid approaches [6]. *Push-based* mechanisms are mostly server-based, where the server informs the caches about updates, whereas *Pull-based* approaches are client-based, where the client asks the server to update or validate its cached data. Finally, in *hybrid* mechanisms the server pushes the updates or the clients pull them.

An example of pull approaches is the time to live (TTL)-based algorithms, where a TTL value is stored alongside each data item d in the cache, and d is considered valid until T time units go by since the last update. Such algorithms are popular due to their simplicity, sufficiently good performance, and flexibility to assign TTL values to individual data items [14], [24]. Also, they are attractive in mobile environments [25] because of limited device energy and network bandwidth [22], [23], and frequent device disconnections [24]. TTL algorithms are also completely client-based and require minimal server functionality. From this perspective, TTL-based algorithms are more practical to deploy and are more scalable.

In this paper, we propose a pull-based algorithm that implements adaptive TTL, piggybacking, and prefetching,

• *The authors are with the Electrical and Computer Engineering Department, American University of Beirut, PO Box 11-0236, Riad El-Solh, Beirut 1107-2020, Lebanon. E-mail: {kmf04, hartail}@aub.edu.lb.*

Manuscript received 22 Jan. 2011; revised 20 Sept. 2011; accepted 25 Jan. 2012; published online 6 Feb. 2012.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-2011-01-0034. Digital Object Identifier no. 10.1109/TMC.2012.37.

and provides near strong consistency guarantees. Cached data items are assigned adaptive TTL values that correspond to their update rates at the data source. Expired items as well as nonexpired ones but meet certain criteria are grouped in validation requests to the data source, which in turn sends the cache devices the actual items that have changed, or invalidates them, based on their request rates. This approach, which we call distributed cache invalidation mechanism (DCIM), works on top of the COACS cooperative caching architecture we introduced in [2]. To our knowledge, this is the first complete client side approach employing adaptive TTL and achieving superior availability, delay, and traffic performance.

In the rest of this paper, Section 2 discusses related work and reveals the contributions of the proposed system, which we elaborate in Section 3. Section 4 provides an analytical analysis of the system, whereas Section 5 presents the experimental results and discusses their significance. Section 6 finishes the paper with concluding remarks and suggestions for future works.

2 Related Work

Much work has been done in relation to cache consistency in MANETs, where the proposed algorithms cover push-, pull-, and hybrid-based approaches.

The work on push-based mechanisms mainly uses invalidation reports (IRs). The original IR approach was proposed in [3], but since then several algorithms have been proposed. They include stateless schemes where the server stores no information about the client caches [3], [4], [16], [17] and stateful approaches where the server maintains state information, as in the case of the AS scheme [20]. Many optimizations and hybrid approaches were proposed to reduce traffic and latency, like SSUM [35], and the SACCS scheme in [18] where the server has partial knowledge about the mobile node caches, and flag bits are used both at the server and the mobile nodes to indicate data updates. Such mechanisms necessitate server side modifications and overhead processing. More crucially, they require the server to maintain some state information about the MANET, which is costly in terms of bandwidth consumption especially in highly dynamic environments. DCIM, on the other hand, belongs to a different class of approaches, as it is a completely pull-based scheme. Hence, we will focus our survey of previous work on pull-based schemes, although we will compare the performance of DCIM with that of our recently proposed push-based approach, namely SSUM [35], in Section 5.

Pull-based approaches, as discussed before, fall into two main categories: client polling and time to live.

2.1 Client Polling

In client polling systems, such as those presented in [19] and [20], a cache validation request is initiated according to a schedule determined by the cache. There are variants of such systems (e.g., [19] and [8]) that try to achieve strong consistency by validating each data item before being served to a query, in a fashion similar to the "If-modified-since" method of HTTP/1.1. In [19], each cache entry is validated when queried using a modified search algorithm, whereas in [8] the system is configured with a probability that controls the validation of the data item from the server or the neighbors when requested. Although client poll algorithms have relatively low bandwidth consumption, their access delay is high considering that each item needs to be validated upon each request. DCIM, on the other hand, attempts to provide valid items by adapting expiry intervals to update rates, and uses prefetching to reduce query delays.

2.2 TTL-Based Approaches

Several TTL algorithms which were proposed for MANETs were motivated by web caches research. These include the fixed TTL approach in [9] and [24] and the adaptive TTL methods in [7], [21], [12], and [26]. Adaptive TTL provides higher consistency requirements along with lower traffic [7], and is calculated using different mechanisms [7], [21], [12], [28], and [29]. The first mechanism in [21] calculates TTL as a factor multiplied by the time difference between the query time of the item and its last update time. This factor determines how much the algorithm is optimistic or conservative. In the second mechanism, TTL is adapted as a factor multiplied by the last update interval. In dynamic systems, such approaches are inappropriate as they require user intervention to set the factors, and lack a sound analytical foundation [11]. In the third mechanism in [37] TTL is calculated as the difference between the query time and the kth recent distinct update time at the server divided by a factor K, and the server relays to the cache the k most recent update times. Other proposed mechanisms take into consideration a complete update history at the server to predict future updates and assign TTL values accordingly [25]. These approaches assume that the server stores the update history for each item, which does not make it an attractive solution. On the other hand, the approach in [30] computes TTL in a TCP-oriented fashion [27] to adapt to server updates. However, it is rather complex to tune, as it depends on six parameters, and moreover, our preliminary simulation results revealed that this algorithm gives poor predictions. Finally, the scheme in [11] computes TTL from a probability describing the staleness of cached documents. At the end, it is worth mentioning that piggybacking was proposed in the context of cache consistency to save traffic. In [10], the cache piggybacks a list of invalidated documents when communicating with the server, while in [36] the server piggybacks a list of updated documents when it communicates with the cache.

TTL-based approaches have been proposed for MANETs in several caching architectures [22], [23], [31], [32], [33], and [15]. The works in [22], [23], and [31] suggest the use of TTL to maintain cache consistency, but do not explain how the TTL calculation and modification are done. A simple consistency scheme was proposed in [32] and [33] based on TTL in a manner similar to the HTTP/1.1 max-age directive that is provided by the server, but no sufficient details are provided. Related to the above, we show in Section 5 that approaches which rely on fixed TTL are very sensitive to the chosen TTL value and exhibit poor performance. In [15], a client prefetches items from nodes in the network based on the items' request rates, and achieves consistency with the data sources based on adaptive TTL calculated similar to the schemes of the Squid

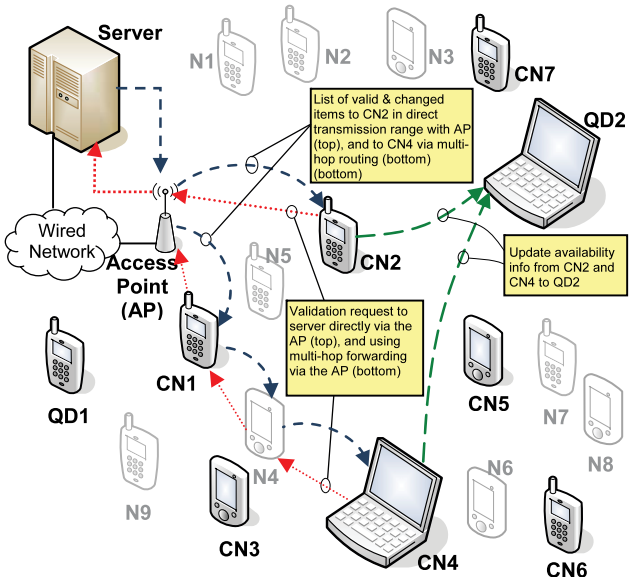

Fig. 1. Overview of DCIM basic design.

cache and the Alex file system. This scheme introduces large traffic, as two invalidation schemes work in parallel, and moreover, the TTL calculations are seemingly inaccurate and are based on heuristics [11].

In summary, the above approaches only provide shallow integration of TTL processing into the cache functionality, and none of them gives a complete TTL-based cache consistency scheme for MANETs. Additionally, they do not include mechanisms for reducing bandwidth consumption, which is crucial in MANET environments.

3 DCIM Architecture and Operations

This section describes the design of DCIM and the interactions between its different components.

3.1 System Model

The system consists of a MANET of wireless mobile nodes interested in data generated at an external data source connected to the MANET using a wired network (e.g., internet) via WiFi Access Points (APs). Nodes that have direct wireless connectivity to an AP act as gateways, enabling other nodes to communicate with the data source using multihop communication. For example, Node CN4 in Fig. 1 is accessing the server through N4 and then CN1, which in turn acts as a gateway by connecting to the Internet via the AP. The data exchanged is abstracted by data items, as mentioned in Section 1.

DCIM is a client-side system that is able to scale to many types of provided services. In fact, DCIM fits more naturally into the current state of the Internet with the prevailing client/server paradigm, where clients are responsible for pulling the data from the server, which in turn maintains little state information and seldom pushes data to them. On the other hand, push-based approaches, like those described in the related work section, rely on the server totally or partially to propagate item changes to the network. This feature though is not implemented in currently deployed application servers, as most application servers do not maintain change history or attempt to calculate expiry times, which is why the client side is depended on to provide such service. We demonstrate later how DCIM performs similar or better than push-based approaches, while keeping all the processing at the client side with little overhead.

The proposed DCIM system builds on top of COACS, which we introduced in [2] and did not include provisions for consistency. For completeness, a description of the COACS operations is provided in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TMC.2012.37. Briefly, the system has three types of nodes: caching nodes (CNs) that cache previously requested items, query directories (QDs) that index the cached items by holding the queries along with the addresses of the corresponding CNs, and requesting nodes (RNs) that are ordinary nodes. Any node, including a QD or a CN, can be a requesting node, and hence, an RN is not actually a special role, as it is only used in the context of describing the system. One, therefore, might view the employed caching system as a two layered distributed database. The first layer contains the QDs which map the queries to the caching nodes which hold the actual items that are responses to these queries, while the second layer is formed by the CNs. The operations of the QDs and CNs are described in more details in Sections 3.4.3 and 3.4.4, respectively.

Finally, it is worth mentioning that although our recently introduced SSUM [35] cache consistency scheme also builds on COCAS, it is a server-based approach, whereas DCIM is completely client-based, introduced to realize the benefits of this class of systems. In this regard, DCIM complements SSUM, which is why we contrast their performance in Section 5 to see how they compare.

3.2 Design Methodology

The goal of DCIM is to improve the efficiency of the cache updating process in a network of mobile devices which cache data retrieved from a data server, without requiring the latter to maintain state information about the caches. The proposed system is pull-based, where the CNs monitor the TTL information and accordingly trigger the cache updating and validation process.

DCIM is scalable by virtue of the CNs whose number can increase as the size of the network grows (each node can become a CN for an item it requests if not cached elsewhere in the network), and thus is more suitable to dynamic MANETs than a push-based alternative since the server does not need to be aware of CN disconnections. DCIM is also more suitable when data requests are database queries associated with tables and attributes. In a push-based approach, the server would have to map a cached query to all of its data sources (table attributes) and execute this query proactively whenever any of the sources is updated. Moreover, DCIM adapts the TTL values to provide higher consistency levels by having each CN estimate the interupdate interval and try to predict the time for the next update and sets it as the item's expiry time. It also estimates the inter-request interval for each data item to predict its next request time, and then prefetches items that it expects to be requested soon.

3.3 DCIM Basic Design

In DCIM, the caching system relies on opportunistic validation requests to infer the update patterns for the data

TABLE 1
Packets Used in DCIM

Packet	Function	Description
CURP	Cache Update Request	Sent from CN to server to validate certain data items
SVRP	Server Validation Reply	Sent from server to CN to indicate which items are valid
SUDP	Server Update Data	Sent from server to CN. It includes updated data items and timestamps

items at the server, and uses this information to adapt the TTL values. These validation requests are essentially requests to the server to refresh a set of data items. The CN polls the server frequently to know about the update times of the items it caches. It also piggybacks requests to refresh the items it caches each time it has reason to contact the server, basically whenever an item it caches expires. Nevertheless, to avoid unnecessary piggybacks to the server, the CN utilizes a two-phase approach. Specifically, at the end of each polling interval (T_{poll}), every CN issues validation requests for the items it indexes that have expired TTLs and have a high request rate. After a configurable number of polling intervals, denoted by N_{poll}, the CN issues a validation request for all the items it caches if at least one item has an expired TTL regardless of its request rate. We refer to the interval $N_{poll} \times T_{poll}$ as the piggyback interval, T_{pigg}. When the server receives a CN's request, it replies with a list of updated as well as nonupdated items. The CN uses this information to adapt the TTL values to the server update rate for each item. The effectiveness of these mechanisms is explained in the experimental evaluation section.

Although in principle it achieves weak consistency, DCIM can attain delta consistency when at least one item has a TTL expired by the end of the piggybacking interval, thus effectively causing validation requests to be issued periodically. Hence, the CN ensures that data items are at most one piggybacking interval stale. Fig. 1 shows a scenario for illustration purposes where two CNs are sending cache validation requests to the server (dotted arrows) via gateway nodes and through the Access Point. The server replies back with lists of valid and changed data items (short-dashed arrows) to the CNs, which in turn update the corresponding QDs asynchronously about the items they cache (long-dashed arrows).

3.4 Detailed Design

In the remainder of this section, we describe the operations of DCIM in details, but first, we list the messages which we added in DCIM (see Table 1) to those already introduced in COACS. The reader can refer to [2] for a complete description of all the COACS messages.

Fig. 2 shows the basic interactions of DCIM through a scenario in which an RN is submitting a data request packet (DRP) for a query indexed in the QD. The QD forwards the DRP to the CN caching the item assuming there was a hit. At the CN, the requested item may be in the waiting list at the moment if it is being validated. Validation requests are

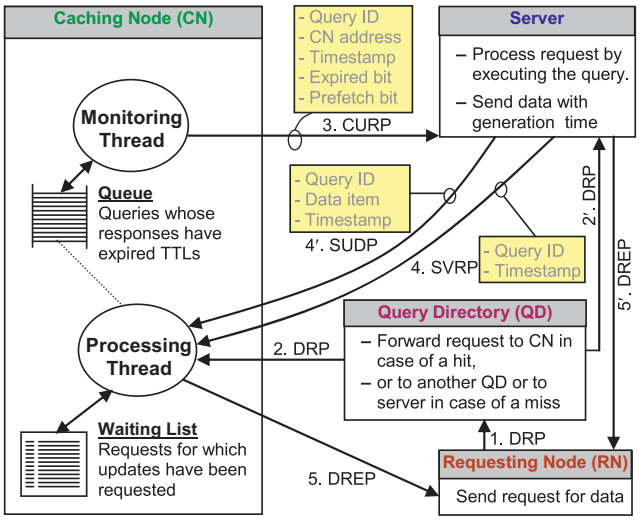

Fig. 2. Interactions between nodes in a DCIM system.

issued by CNs using CURP messages. Each entry in the message consists of the query associated with this item, a timestamp (last modification time), a "prefetch" bit (if set, instructs the server to send the actual item if updated), and the "expired" bit (tells if an item is expired). Upon receiving a CURP message, the server identifies items that have changed and those that have not, and sends the corresponding CNs in SVRP messages the ids of items that did not change and those that changed but do not have the prefetch bit set. It also sends the CNs SUDP messages containing the actual items if they were prefetched by the same CN and have changed. Now the CN releases the request from the waiting list and sends the updated cached response to the RN via a data reply (DREP) message.

3.4.1 TTL Adaptation

The CN in DCIM has a partial picture about the update pattern of each item at the server using the piggybacking mechanism. It stores the last update time of each item from the last validation request, and uses this information to predict the next update time. However, the CNs are after all mobile devices which have constraints in terms of power, processing, and storage capabilities, and obviously, sophisticated prediction schemes are slow and inadequate to use in this context. Alternatively, we use a running average to estimate the interupdate interval, using timestamps of the items from the server's responses to issued validation requests. The CN can then calculate its own estimation for the interupdate interval at the server, and utilizes it to calculate the TTL of the data item. This is an exponentially weighted moving average, and has the form: $IUI(t) = (1-\alpha) \times IUI(t-1) + \alpha \times LUI$, where $IUI(t)$ is the estimated interarrival time at time t and LUI represents the last interupdate interval. The CN only needs to store the estimated interval and the last updated time. In fact, this method has many properties that make it suitable for usage in this situation, mainly because of its simplicity and ease of computation, the minimum amount of data required, and the diminishing weights assigned to older data [34]. There are two parameters that control this estimator which are the

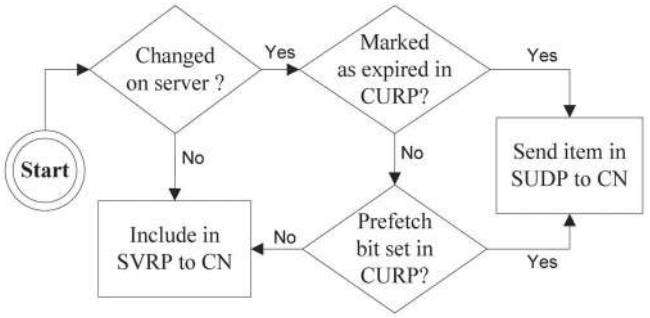

Fig. 3. Decision flow at the server.

initial value $IUI(0)$ and the value of α, whose value should be small, i.e., between 0.1 and 0.2, as to minimize the effect of random fluctuations, even if it means larger convergence times [26] (proven in Appendix B, which is available in the online supplemental material). In the simulations we describe later, α was set to 0.125 and $IUI(0)$ to 0. In Section 3.5.4, we show how this TTL calculation scheme is incorporated in DCIM.

3.4.2 Server Operations

As this approach is client-based, the processing at the server is minimal. When the server receives a CURP message from the CN, it checks if all items have been changed by comparing their last modified times with those included in the request. Items that have not changed are considered valid, and their ids are included in the SVRP response to the CN. On the other hand, items that have changed are treated in two ways: Expired items (those having the expiry bit set in the CN validation request) as well as nonexpired ones but having the prefetch bit set are updated by sending SUDP packets (which contain the actual data items and the associated timestamps) to the originating CNs. As to the items whose expiry and prefetch bits are not set (i.e., will not be requested soon), the server informs the CN about them using an SVRP message. This is summarized in the flow diagram of Fig. 3. As such, the server only reacts to the received CURP messages that do not require it to maintain any state information, and thus it does not need to be aware of the MANET dynamics, including any CN or QD disconnections. Given this, it is not difficult to deploy DCIM in an Internet environment. For example, DCIM can be suited to be deployed on top of HTTP through mapping DCIM's request directives into HTTP header fields and utilizing extended headers, as allowed by HTTP/1.1 and defined in RFC 2616. The details of suiting DCIM to function on top of HTTP/1.1 or any other protocol are, however, out of scope of this paper.

3.4.3 QD Operations

In contrast to the CNs that become caching nodes when they first request noncached data, QDs are elected based on their resource capabilities, as described in [2]. A procedure is included in [2] that explains how the number of QDs in the system is bounded by two limits. The lower bound corresponds to having enough QDs, such that an additional (elected) QD will not yield an appreciable reduction in average QD load. The upper bound, on the other hand, corresponds to a delay threshold, since traversing a larger number of QDs will lead to higher response times. Between these limits, the number of QDs can change dynamically depending on how much of the QD storage capacity is used. In the simulations performed in this work, the number of QDs averaged 7 at steady state when the number of nodes was 100.

Concerning load, it was shown in [2] that the average load on a QD node is 1.5 times the average load on a CN node. Since in DCIM the QDs are not assigned additional roles as compared to [2], their average load should not change. In general, the QD system yields a high hit rate, which causes the request to traverse less QDs on average, and consequently keeps the load per QD reasonably low. Moreover, other than the occasional COACS updates about replaced items (due to capacity constraints), the CNs never update the QDs about the expired data items. Consequently, the QD always forwards the DRP to the CN in case of a hit. This makes the system simpler and saves traffic, but might incur additional delay only if the item is expired at the CN, as this will cause the CN to contact the server to update the data item. Nevertheless, this additional internal delay is small when compared to the server delay, and it is compensated by reduced consistency updates from the CNs to the QDs. Also, we show later in the experimental evaluation section that the system maintains an acceptable hit rate, meaning that the probability a data item will be expired at the CN is low.

TABLE 2
Elements of the General Cache Information Table

Parameter	Description
CN_{ID}	Identifier of this CN
WL	Query processing waiting list

3.4.4 CN Processing

The CNs store the cached queries along with their responses plus their IDs, and the addresses of the QDs indexing them. They are distributed in the network and cache a limited number of items, which makes monitoring their expiry an easy task. A CN maintains two tables to manage the consistency of the cache: the *Cache Information Table* whose data is common to all queries whose responses are locally cached (Table 2), and the *Query Information Table* that stores query-specific data (Table 3). As shown, the CN maintains the weighted average of inter-request interval (IRI) for each data item it holds (in a manner similar to the

TABLE 3
Elements of the Query-Specific Cache Information Table

Parameter	Description
q_{ID}	Identifier of the locally cached query
QD_{ID}	Identifier of the QD that indexes this query
Timestamp	The last modified time of the data item
TTL	Time to live value associated with this query
IRI	The estimated inter-request interval for this query
IUI	The estimated inter-update interval for this query
State	Tells if the item is expired or issued for validation

computation of the interupdate interval). The process that runs on the CN includes two threads: a monitoring thread and a processing thread.

Monitoring thread. The monitoring thread checks for expired data items, issues validation requests, and requests updates for data items. It performs these operations in two functions.

Inner loop function. After the CN sleeps during the polling interval (T_{poll}), it iterates over the entries corresponding to the data items it caches, checking each item's TTL value. If an item has an expired TTL, the CN sets its expiry bit and its state to *INVALID*. It also sets its "prefetch" bit if the item is predicted to be requested before the end of the current piggybacking interval, i.e., if the last requested time plus the average inter-request interval is lower than the end of the current piggyback period ($N_{poll} \times T_{poll}$), meaning that in this case the item will be requested with high probability by one or more RN nodes before the end piggybacking interval. In what follows we refer to items that are predicted to be requested before the end of the current piggybacking interval as items with high request rate, while other items are referred to as items with low request rate. The CN then sets the state field to *TEMP_INVALID* to indicate that a validation request for the item is in progress. Normally, nodes that request invalidated data items will have to wait till the server updates the CNs with new versions upon their request. At the end of the inner loop, the CN prepares a CURP, and includes in it the validation requests for items that have expired and whose prefetch bits are set.

Outer loop function. When the monitoring thread completes N_{poll} iterations (i.e., after a piggyback interval, defined above), the CN checks if at least one item has expired. If so, it issues a validation request for the whole collection of cached items stored at the CN. In this request, similar to the one issued in the inner loop, a prefetch bit indicates if the item is expected to be requested soon, as was described above. If it is set, the server sends the actual item, else, it just invalidates it. Hence, the outer loop allows the CN to piggyback validation requests for all items when there is a need to contact the server.

Note that the inner loop function issues validation requests *only* for expired items having high request rates (items predicted to be requested before the end of the piggybacking interval as specified before), and updates them if necessary. Expired items with low request rates have to wait for at most $N_{poll} \times T_{poll}$ to be validated, while those with high request rates wait for at most T_{poll}. Since prefetching data items to save on query delay constitutes most of the traffic consumption in the network, prefetching only highly requested items helps to limit the bandwidth consumption in the network. In this regard, we note that in delay intolerant networks, the "prefetch" bit can be set for each item regardless of its request rate, assuming it was requested at least once in the past. This way, all items will be prefetched and the hit rate will be forced to be 100 percent or very close to it (when accounting for items that are requested while being validated), thus reducing response time considerably. Fig. 4 summarizes the operations of the inner loop and outer loop functions.

Processing thread. This thread handles data requests from RNs and replies from the server (i.e., SUDP and SVRP packets) in response to CURP messages, and computes the TTL value.

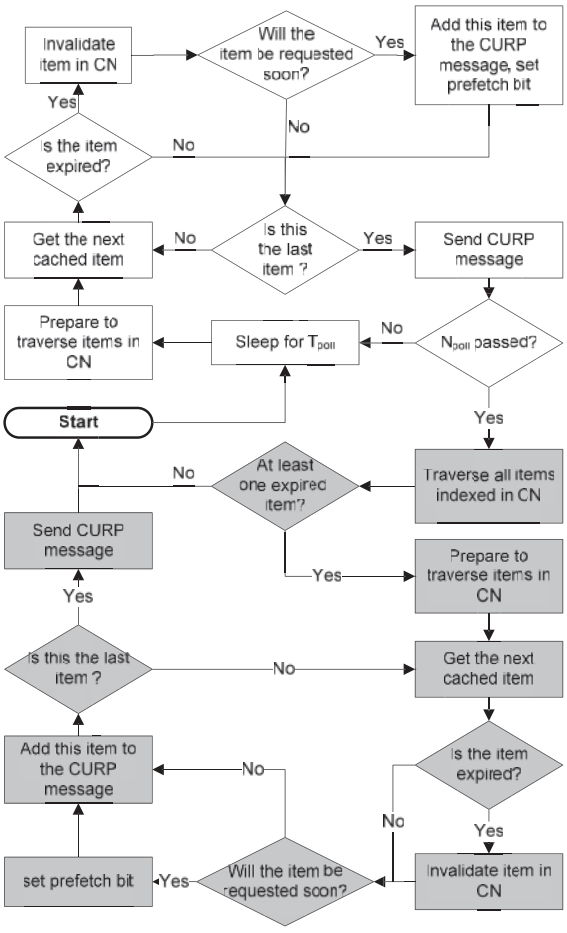

Fig. 4. Inner loop and outer loop (shaded part) functions.

Processing data request messages. The CN checks the requested item in the DRP, and if it is *INVALID*, it issues an update request directly to the server, changes its state to *TEMP_INVALID*, and places the query on a waiting list. In the meanwhile, if the CN gets a DRP for the same item before the server replies, it also puts it on the waiting list. In all other cases, the query is processed by sending the requested item to the RN via a DREP message.

Processing SVRP and SUDP messages. If an SUDP packet was received, it must be for an item that has changed at the server. The CN calculates its TTL as explained below, and if the SUDP makes reference to items that have requests placed in the waiting list, those items are sent to the corresponding requesting nodes. On the other hand, the SVRP is sent from the server in response to a CURP packet, and it is expected to only contain the ids of the items that did not change, and those of the items that changed but were marked as unexpired and had the prefetch bit not set in the CURP (illustrated in Fig. 3). The CN updates the TTL of all elements whose ids are contained in the SVRP. It helps to reiterate here that there are items which were specified in the CURP packet but not sent as part of the SVRP because the actual updated data items were sent to the CNs as part of the SUDP message.

TTL calculation. In DCIM, the exact TTL calculation performed by the CN depends on whether the item was expired at the server or not, which is information contained

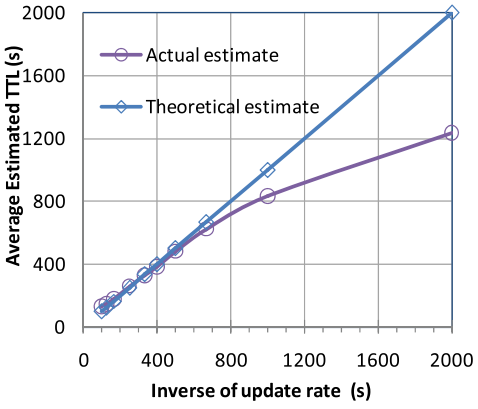

Fig. 5. Average TTL versus inverse of interupdate interval.

in the SVRP and SUDP messages. The TTL value is calculated as per the steps below.

- If the item has changed on the server, the SVRP would contain the last updated time (denoted by LU_{new}), given the item had the prefetch bit not set, whereas the SUDP would contain the same value if this bit was set. In both cases, TTL is set to $(1-a) \times IUI + a \times (LU_{new}\text{-}Timestamp)$.
- If the item did not change on the server and the TTL did not expire on the CN, the TTL will not be modified. This case occurs because of the piggybacking procedure described before.
- If the item expired on the CN, but did not change on the server, the CN increases the TTL value by considering the current time as the update time, without changing the timestamp value it stores. The TTL value will be set to $(1-a) \times IUI + a \times (CurrentTime - Timestamp)$.

In some cases, the actual interupdate interval at the server could increase while the estimated interupdate interval may not have updated yet. This causes the last calculated interupdate interval when the item was last changed to become shorter than the time elapsed since the past update. This gives rise to a next expiry time occurring in the past. Should this situation occur, the CN reacts by setting the next expiry time to the estimated interupdate interval added to the current time (the time the item was validated when its timestamp did not change, or changed but the change was too old). This is done by setting TTL to *CurrentTime - Timestamp +IUI*. This situation stays in effect until the item gets a new timestamp (changes on the server).

For illustration purposes, a sample plot for the TTL value versus the update rate of a Poisson update process is shown in Fig. 5. It shows that at very low update rates (less than 1 update per 1,000 s), the estimated TTL does not adapt well. However, in actuality, time goes beyond the 2,000 s considered for this simulation time, meaning that more item updates will occur on the server during the longer time interval. It follows that the actual TTL will not diverge to the same extent as shown in Fig. 5. This shows the effectiveness of the update rate adaptation mechanism in calculating TTL values. This is further elaborated in the experimental evaluation section where the data consistency is shown to be near 1.

3.4.5 Handling CN and QD Disconnections

It is fair to assume that CNs and QDs will go offline from time to time either temporarily or permanently. In either case, DCIM should react efficiently to keep the system running: it does not attempt to proactively account for CN or QD disconnections, but rather, it reacts to these events by relying on the QDs to detect when CNs go offline. In case of a query hit, the QD will always try to forward the data request to the CN, and consequently any routing protocol will return a route error if no route can be established to the CN. This will indicate that the CN is not reachable or equivalently disconnected. In such a case, the QD instructs the RN to request the item from the external data source as it would do in a case of a data miss. As a result, the RN would become a CN for this item in particular, and the QD will mark this entry as invalid. All the QDs behave similarly for each request they receive for items cached in offline CNs. When a CN rejoins the network, it broadcasts a HELLO message as any new node joining the network would do [2]. The QDs react by sending this CN an entry deletion packet (EDP) which applies to any item cached in this CN and marked as invalid because it was requested during the CN's disconnection and consequently cached somewhere else. The CN will remove this item and resumes its operations normally for the other items.

On the other hand, the QDs can disconnect but with a lower probability than the CNs since they are limited in number in the network and are more capable nodes in terms of connectivity and battery. Moreover, a QD disconnection is less severe than that of a CN. Rebuilding a CN's cache requires communication with the QDs and with the server to fetch the actual items, in addition to the loss of the associated consistency information. Alternatively, the reconstruction of the QD cache only requires communications with the CNs. A QD disconnection is easily detected by either an RN or a QD in the system during a data request. When an RN or a QD attempts to forward a DRP to a disconnected QD, the routing layer will return a routing error. From here after, the node that first detects a disconnected QD will initiate a QD recovery procedure as described in [2]. Briefly, this procedure includes the election of a new QD, informing the network about it, and then reconstructing its cache from the CNs in the network. Needless to say, if a previously offline QD rejoins the network; it drops its QD role, unlike a rejoining CN, as was described earlier.

With regard to these procedures, the performance of the system could be improved, through incorporating into the design a replication scheme, similar to the one in [43] and [44], to replicate data on both the QDs and CNs and hence, reduce the overhead associated with node disconnections, especially the query delay that results from the maintenance process.

4 Analysis

We analyze DCIM to assess the bandwidth gain and the query response time gain as compared to the poll-every-time (PET) and push-based stateful (PBS) consistency schemes. We define the bandwidth gain as the difference between the amounts of PET and PBS traffic, on one hand, and DCIM traffic on the other hand. Similarly, the query

response time gain is the difference between the times it takes to get the answer of the query (measured from the time of issuing the query). The results are in agreement with the results shown in Section 5.

Requests for data within the network and arrival of data updates at the server are assumed to be random homogenous Poisson processes, and thus the interarrival times are represented by exponential random variables, as was suggested in [14] and [42]. We use λ_R to denote the rate of requests and λ_U for the item update rate, and suppose that each query or data item can have its own rate. The PDFs of the interarrival times are therefore

$$P_R(t) = \lambda_R e^{-\lambda_R t}, P_U(t) = \lambda_U e^{-\lambda_U t}. \quad (1)$$

To estimate the response time and traffic gains, we borrow concepts from our previous work in [2] related to the average number of hops required in the various situations in the calculations.

- H_C is the average number of hops between the corner of the topology and a randomly selected node. It is used when a packet is sent between the server and a node in the network.
- H_R is the expected number of hops between any two randomly selected nodes.
- H_D is the expected number of hops to reach the QD containing the reference to the requested data, in the case of a hit.
- T_{in} is the transmission delay between two neighboring nodes (i.e., one hop delay), while T_{out} is the round trip time between the MANET and the server.
- S_D is the size of the data packet and S_R is the size of the request.

In what follows, we discuss the time response and bandwidth gains for DCIM when compared with the PET and PBS schemes. We show that although a push-based stateful scheme could offer slightly smaller response times, it generates considerably more traffic.

4.1 Response Time Gain

In Appendix C, which is available in the online supplemental material, we derive the response time gain of DCIM over PET and over PBS, and show them in (2) and (3):

$$GT_{PET} = T_{RTT} - P_{SC1} \times T_{RTT} - (1 - P_{SC1}) \times T_{MAN}, \quad (2)$$

$$GT_{PBS} = -P_{SC1} \times T_{RTT} + P_{SC1} \times T_{MAN}, \quad (3)$$

$$\text{where} \quad T_{RTT} = T_{out} + T_{in}(2H_C + H_D), \quad (4)$$

$$T_{MAN} = T_{in}(2H_R + H_D), \quad (5)$$

$$P_{SC1} = \lambda_U \times T_{RTT} \times e^{-\lambda_U \times T_{RTT} - 1}. \quad (6)$$

The gains are plotted in the left graph of Fig. 6, where values consistent with the corresponding values in the simulations were used: $H_D = 5, H_R = 5.21, H_C = 5.21, T_{in} = 5$ ms, $T_{out} = 70$ ms, and $\lambda_U = 1/500$. As implied from (2), the gain mainly depends on the update rate, which causes it to decrease slightly when it increases. We note that for PBS, we consider the best case scenario, where items are always up-to-date and answered directly from the CNs. In the graph, the lower curve confirms that in DCIM the majority of the requests are answered from the MANET, this is why the time response difference is less than 10 ms.

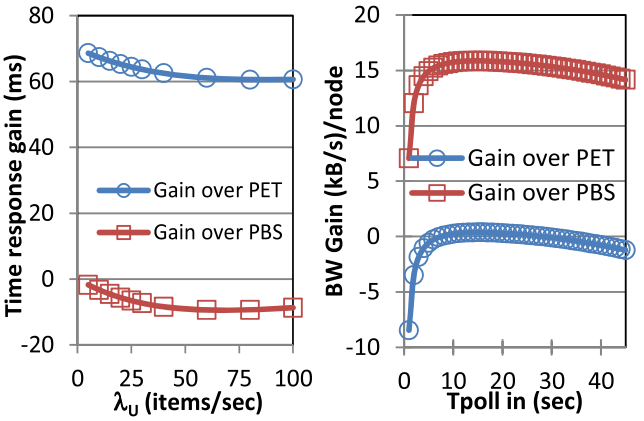

Fig. 6. Average time response gain (left) and bandwidth gain (right).

4.2 Bandwidth Gain

The expressions for the bandwidth consumption of PET, PBS, and DCIM are derived in Appendix D, which is available in the online supplemental material, and are shown below in (7), (8), and (9), respectively,

$$R_{tot} \times (P_{poll} \times B_{pi} + (1 - P_{poll}) \times B_{po}), \quad (7)$$

$$B_{rej} + B_{rej-rq} + B_{rej-up} + B_{disc} + B_{upd}, \quad (8)$$

$$M \times (B_{Rpoll} + B_{poll\,nc} + B_{pollc}) - B_{Rpigg} - B_{piggnc} - B_{piggc}. \quad (9)$$

In the expressions above, $B_{po} = S_R(H_D + H_C + H_R) + S_D H_C$, $B_{pi} = S_R(H_D + 2H_C)$, $R_{tot} = \lambda_R \times T_{pigg}$, $T_R = 1/\lambda_R$ and T_{pigg} is the piggybacking interval. $P_{poll} = e^{-\lambda_U \times T_R}$, $B_{pollnc} = (K-l) \times S_R H_C$, $B_{pollc} = l \times (S_D H_C + S_R H_R)$, $B_{Rpoll} = K \times S_R H_C$, $B_{piggc} = m \times S_D H_C$, $B_{piggnc} = (N-m) \times S_R H_C$, $B_{Rpigg} = N \times S_R H_C$, and N is the number of items. $B_{rej} = E[N_{disc}] \times p_r \times H_c S_u$, $B_{rejrq} = D \times q \times \lambda_R \times E[N_{disc}] \times p_r \times (H_c S_R + H_c S_D + H_c S_u)$, $B_{rej-up} = D \times q \times (1 - \lambda_R) \times \lambda_U \times E[N_{disc}] \times p_r \times (H_c S_R + H_c S_D + H_c S_u)$, $B_{disc} = q \times \lambda_R \times E[N_{disc}] \times (1 - p_r) \times (H_c S_R + H_c S_D + H_c S_u)$, $B_{upd} = \lambda_u \times N \times H_c S_D$. $E[N_{disc}]$ is the CN disconnection rate, q is the average number of data items in CN, p_r is the rejoining rate, D is the disconnection interval, and S_u is the control message size. The terms for l, K, and m, are found in Appendix D, which is available in the online supplemental material.

The bandwidth gain is plotted in the right graph of Fig. 6, where in addition to the same hop count values as those utilized above, the following values are used $\lambda_R = \lambda_U = 1/500$, $N = 4{,}000$, $M = 20$, $S_R = 0.5$ KB, $S_D = 10$ KB, $S_U = 0.15$ KB, $q = 20$, $E[N_{disc}] = 15$ (highly dynamic scenario), and $D = 20$ s. It is worth noting, that N represents the number of cached items (requested at least once before), rather than the total number of items; this matches the experimental results since not all items will be requested within the simulation time. In effect, the traffic resulting from large piggybacking intervals is lower than that of small piggybacking interval. Also, the traffic demands for DCIM decrease exponentially for small polling intervals in

both the analytical and experimental results. It is evident that DCIM is less costly than PBS, which relies on maintaining a full state of the cache system at the server and pushing the updates to the CNs.

This traffic analysis shows that the polling interval is the main parameter that affects performance, and thus it needs to be tuned. Given the traffic expression, the optimal polling interval is the value that results in the lowest traffic consumption (equivalent to the highest bandwidth gain since poll every time does not depend on the polling interval). One can use a numerical method or plot the traffic versus polling interval (Fig. 6) to find the optimal polling interval which is in the vicinity of 15 s.

5 Experimental Results

DCIM was implemented using ns2 [36], and a new database class was developed that mimics the server process in storing and updating data items and in processing the validation requests. Timers were utilized to implement the monitoring thread: the timer sleeps for the polling interval duration and then wakes up to run the innerloop function, i.e., after N_{poll} runs of the innerloop, (the piggybacking interval) the outer-loop is invoked. Ns2 is a single threaded simulator, but it is nevertheless capable of controlling the operations of the timers autonomously, thus acting similar to a multithreaded application.

Two additional schemes were implemented for comparison: the poll-every-time mechanism (considered in Section 4), where each time an item is requested, it is validated; and the fixed-TTL mechanism, where all items have the same expiry interval. The TTL value is calculated by adding to the current time the expiry interval, and when a TTL value expires, the item is flagged as such, and is fetched from the server whenever it is requested.

In order to assess the effectiveness of piggybacking, request rate adaptation, and polling interval mechanisms of DCIM, we additionally implemented three versions of DCIM, each of which had one of the above mechanisms removed. The first has piggybacking disabled so that only expired items are validated in both the innerloop and outerloop functions. The second has the "prefetch" bit always set for all requests so all items are prefetched from the server regardless of their request rate. The last version only implements the update rate adaptation mechanism, where items are validated when they expire.

The simulation area was set to $400 \times 400\ \mathrm{m}^2$, populated with 100 nodes that were randomly distributed. Propagation was according to the two-ray model, and the node's bitrate was set to 2 Mbps. Mobility was based on the random waypoint model (RWP), with a maximum speed of 2 m/s. The server node was connected to the MANET via a gateway and a wired link whose propagation delay was simulated at 40 ms, thus resulting in a server access delay of 80 ms. The server had 10,000 items which were updated according to a Poisson random process at an average rate of about 20 items/s. In the default scenario, each node issues a data request every 10 s according to a Zipf access pattern, frequently used to model non-uniform distributions [38]. In Zipf law, an item ranked $i(1 \leq i \leq nq)$ is accessed with probability: $1/(i^{\theta} \sum_{k=1}^{n_q} 1/k^{\theta})$, where θ ranges between 0 (uniform distribution) and 1 (strict Zipf distribution). The default value of the Zipf parameter θ was set to 1. In the default scenario, the capacity for each of the caching nodes (CNs) is 200 Kb. The simulation parameters are summarized in Table 4.

TABLE 4
Summary of the Default Simulation Parameters

Simulation Parameter	Default Value	Simulation Parameter	Default Value
Simulation time	2000 sec	Size of data item	10 KB
Network size	400×400 m^2	Number of data items updated/sec	20
Wireless bandwidth	2 Mb/s	Delay at the data source	40 ms
Node trans. range	100 m	Node request period	10 sec
Num. of nodes	100	Node request pattern	Zipf (θ=1)
mobility model	RWP	Node caching capacity	200 KB
Node speed (v)	2 (m/s)	Cache Replacement	LRU
Node pause time	30 sec	Polling interval	4 sec
Num. of data items	10,000	N_{poll}	20

In order to calculate the number of runs required for achieving 10 percent confidence interval with an acceptable significance level of 95 percent, we ran a scenario with default parameter values 10 times. For each simulation run, the pseudorandom generator seed (based on the clock of the ns2) and the node movement file were changed. The consistency ratio, delay of the system, and the average traffic per node were computed starting from $T = 500$ s, and the mean plus standard deviation for each set were calculated. Next, the number of runs was computed using the central limit theorem, as discussed in [1]. The error values for the consistency, delay, and traffic were chosen as 0.05, 2 ms, and 0.3 kbps, respectively. The required number of runs was found to be equal to 5 for the consistency ratio, 5 for the delay, and 6 for the traffic. We therefore computed the result of each scenario from the average of six runs.

The reported results are from 5 experiments that involve varying the request rate, the update rate, the item popularity, the maximum velocity, the polling interval, and the data item size. The results are the 1) consistency ratio, 2) query delay (regardless of the source of the results), 3) cached data query delay, 4) uplink traffic, 5) downlink traffic, and 6) average overhead traffic.

5.1 Varying the Request Rate

In this experiment, the inter-request interval was varied between 5 s and 120 s, and the results are plotted in the graphs of Fig. 7. It is evident that poll every time provides the highest consistency ratio (top left graph), since the requested items are always validated, and the items are always fresh, except in certain cases when they change just after being validated. However, when using fixed TTL, the caches might serve stale items (as in the case of $\mathrm{TTL} = 500$ s), but this possibility decreases when the TTL is less than the update interval (as in the case of $\mathrm{TTL} = 100$ s). As a matter of fact, getting the right TTL value is a key issue in relation to the performance of client-based consistency approaches. DCIM is a better approach as it tries to get the appropriate TTL value through piggybacking, which helps in getting a high consistency ratio. Moreover, prefetching

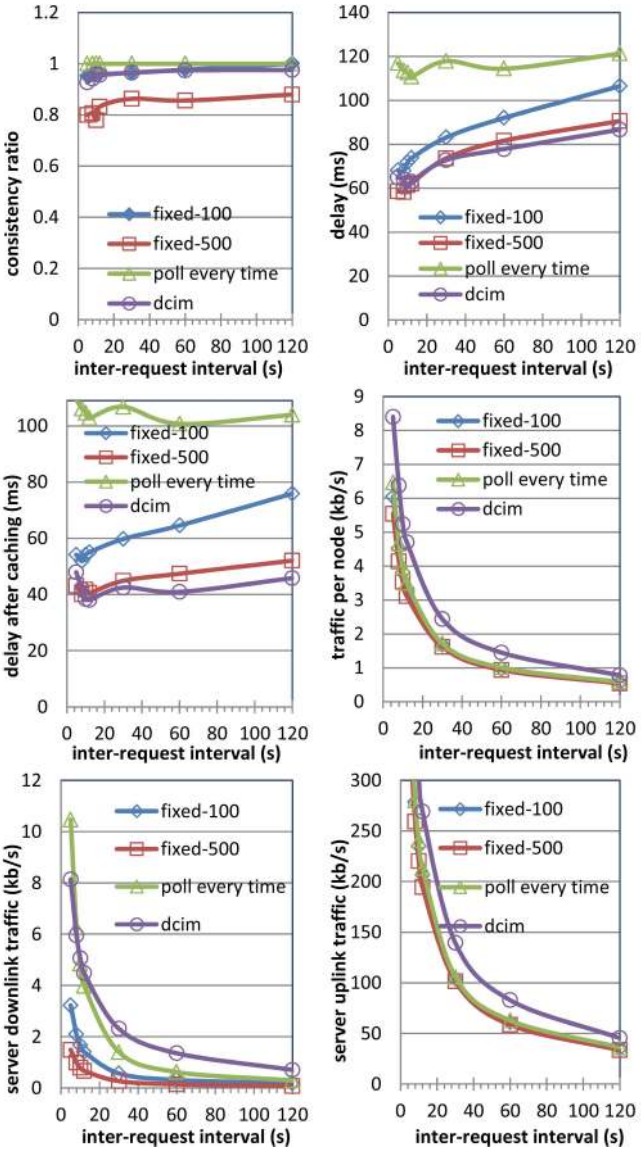

Fig. 7. Performance measures versus inter-request times.

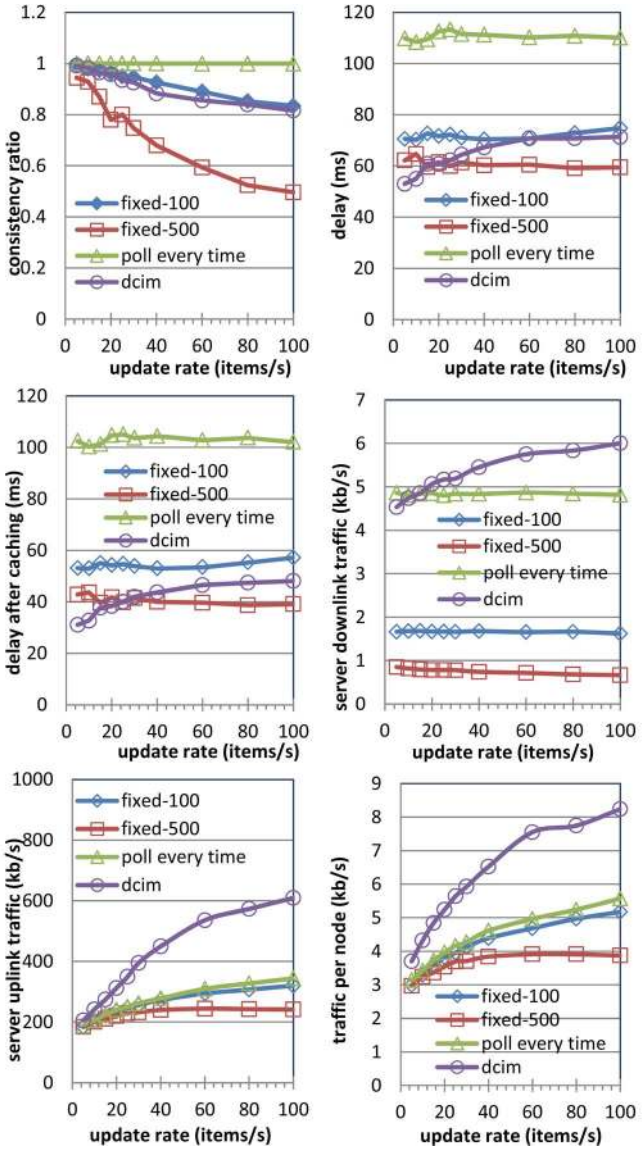

Fig. 8. Performance measures versus update rates.

enables DCIM to provide a high hit ratio, and hence much lower delays than the other approaches. As also shown, the query delay gets smaller after the item is cached but increases by a small margin due to less prefetching as the request rate decreases. Finally, DCIM consumes more traffic on the server side due to prefetching, but is not far off from the other schemes. As for the node traffic, by piggybacking large amount of items, DCIM consumes more traffic when compared to other approaches. However, as the request rate decreases, prefetching does not happen that often, and this leads to lower traffic as shown in the graph. This is how DCIM adapts prefetching to the request rate of items.

5.2 Varying the Update Rate

The results for this scenario are shown in Fig. 8. A TTL value of 100 s is less than the interupdate intervals in all of the scenarios simulated, and hence, it must provide the best consistency level. As shown, DCIM's consistency ratio coincides with that of $\mathrm{TTL} = 100$ s, which is higher than that of $\mathrm{TTL} = 500$ s. Of course, increasing the update rate in any TTL algorithm would decrease its consistency, but with a good TTL estimate, an acceptable consistency could be obtained (comparing $\mathrm{TTL} = 500$ s and DCIM at 100 update/s). Nevertheless, fixed TTL approaches have higher hit rates than poll every time, but less than DCIM, which uses prefetching. This implies that the delay after caching for DCIM is the lower than that of polling every time and fixed-100, but it may exceed that of fixed-500 that keeps the element for a longer time.

The gains in delay and consistency, discussed above, are manifested in a modest increase of traffic as the update ratio increases. However this traffic is not high at the server, and is very low in the MANET (less than 10 kbps, while the bandwidth is 2 Mbps). The reason for the traffic increase is the piggybacking of requests, which increases in frequency as update rates increase. Without this traffic though, the CNs would not be able to infer the update rate and calculate reliable TTL estimates.

5.3 Varying the Zipf Parameter

Varying the θ value effectively varies the popularity of the data items, and is analogous to varying the items' request

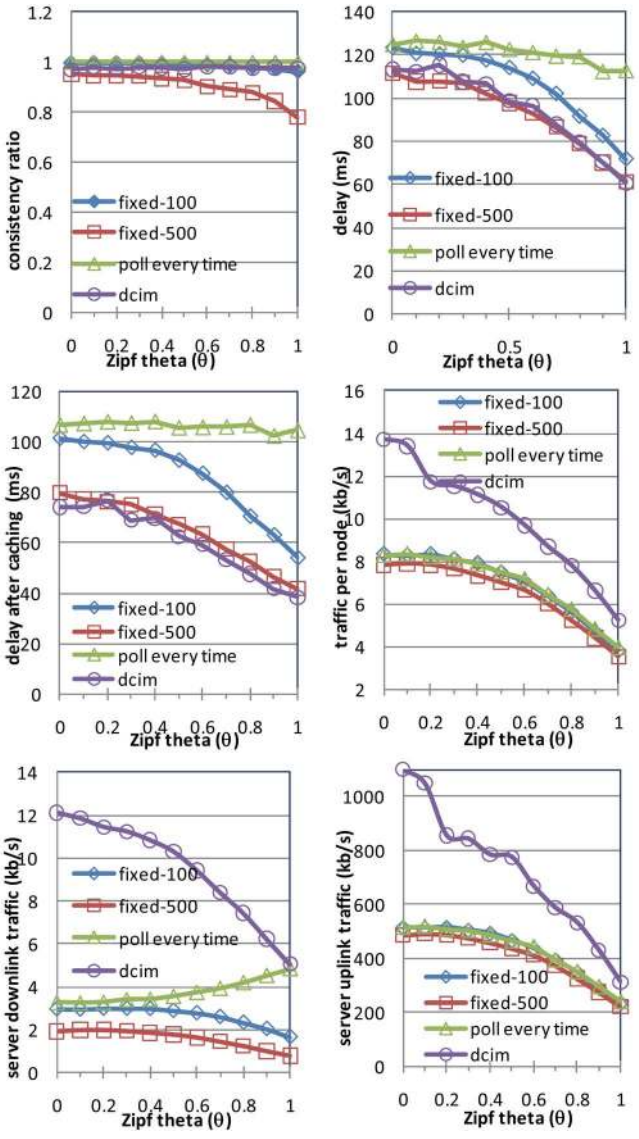

Fig. 9. Performance measures versus the zipf parameter.

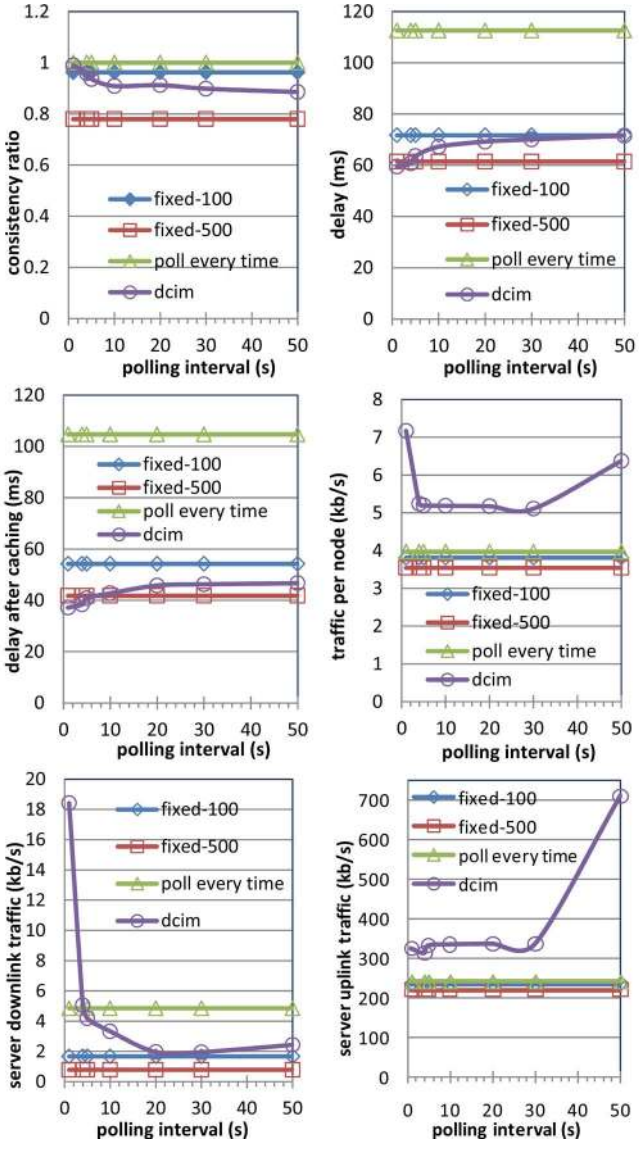

Fig. 10. Performance measures versus the polling interval.

rates. This scenario actually shows the prefetching adaptation to the request rates. As θ increases, the diversity of the requested items decreases, meaning that a smaller subset of the items is requested more. In case one item is updated at the server before the TTL expiration, more stale cached items will result. As seen in Fig. 9, this is why the consistency ratio decreases as θ increases. However, DCIM maintains the TTL for all items regardless of their request rates, and this gives a constant consistency at 98 percent. The situation is reversed when considering hit ratios. For low θ values the hit ratio for fixed TTL is low since requests are distributed across all items, which increases the probability of expired items while the request interval is fixed. As θ increases, the requests will be distributed over a smaller set which increases the probability of hits. It is evident that through prefetching, DCIM provides nearly constant hit rate, which lowers the delay as explained before. DCIM produces more traffic when compared to the other approaches, but this decreases as θ increases since more items will have lower update rates, and will therefore not be validated as frequently.

5.4 Varying the Node Velocity

The maximum node velocity was varied between 0 m/s and 20 m/s, and the obtained results showed that velocity changes produced no special outcome. There was, however, a mild increase in the delay, which is considered normal. In fact, the use of a proactive routing protocol in these simulations masked the delay by making the paths always available. We do not show the relevant graphs in this paper due to the above and to space restrictions.

5.5 Varying the Polling Interval

The polling interval is increased from 1 to 50 s, while the fixed TTL values are kept constant, i.e., 100 and 500 s. The results are shown in the graphs of Fig. 10. This increase caused a mild decrease in the consistency ratio and the hit ratio, and consequently, an increase in the delay, which remains below that of fixed TTL (100 s) and polling every time. Also, the traffic in the uplink direction increases when the piggyback interval increases due to the decrease of hit rate. Finally, it is worthy to point out that by increasing the

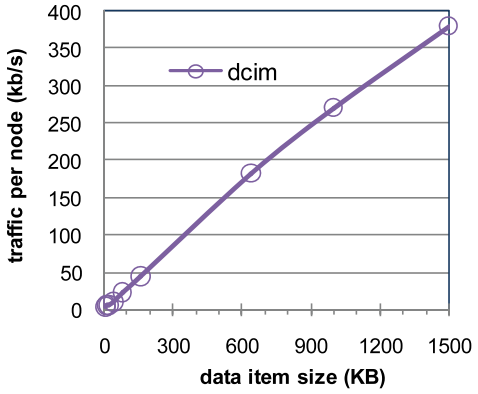

Fig. 11. Effect of data item size on generated traffic.

polling interval, the validation requests from the inner loop function become farther apart in time. However, when the piggybacking interval is very large, the CN will predict that items will be requested before the end of this interval, which leads to more prefetching and consequently more traffic.

5.6 Varying the Data Item Size

In this experiment, the size of the data item was varied between 5 KB and 1,500 KB. The results are shown in Fig. 11, where it is evident that DCIM at high data sizes (1,500 KB) still demands less traffic than the network bandwidth (2 Mbps) leaving room for other applications to operate. Also, it is evident that the traffic per node increases linearly with the increase in data item size as it should be theoretically. This shows that there are dropped packets, which is reflected in the ratio of accepted requests (not shown for size considerations) which does not change as the data item size increases.

5.7 Effectiveness of DCIM Mechanisms

To demonstrate the effectiveness of each individual mechanism of DCIM, as was mentioned at the start of this section, different versions of DCIM were developed and simulated. Fig. 12 shows the obtained results.

One can easily observe that DCIM achieves better consistency over its variants. This illustrates that piggybacking offers an added value by providing a better estimate of TTL, which results in more accurate estimates of expiry times, and consequently higher data consistency. However, piggybacking induces more overhead traffic, resulting from the validation requests and prefetching. We only show the uplink server traffic as it is more critical than the downlink traffic. The server uplink traffic for DCIM is slightly higher than the "pigg-disabled" scheme. However, DCIM traffic would have been much larger if the request rate adaptation scheme was not implemented as seen in the "req-disabled" version. Thus, prefetching highly requested items saves on traffic and provides acceptable query delay as seen in Figs. 7, 8, 9, and 10. Moreover, as the polling interval increases, the effect of piggybacking decreases so that DCIM and "pigg-disabled" converge in performance. As for the "poll-disabled" mechanism, it shows the lowest data consistency in all the graphs, and higher traffic overhead than DCIM. This shows the appropriateness of the proposed mechanisms. Piggybacking increases data consistency and prefetching adapted to the request rate

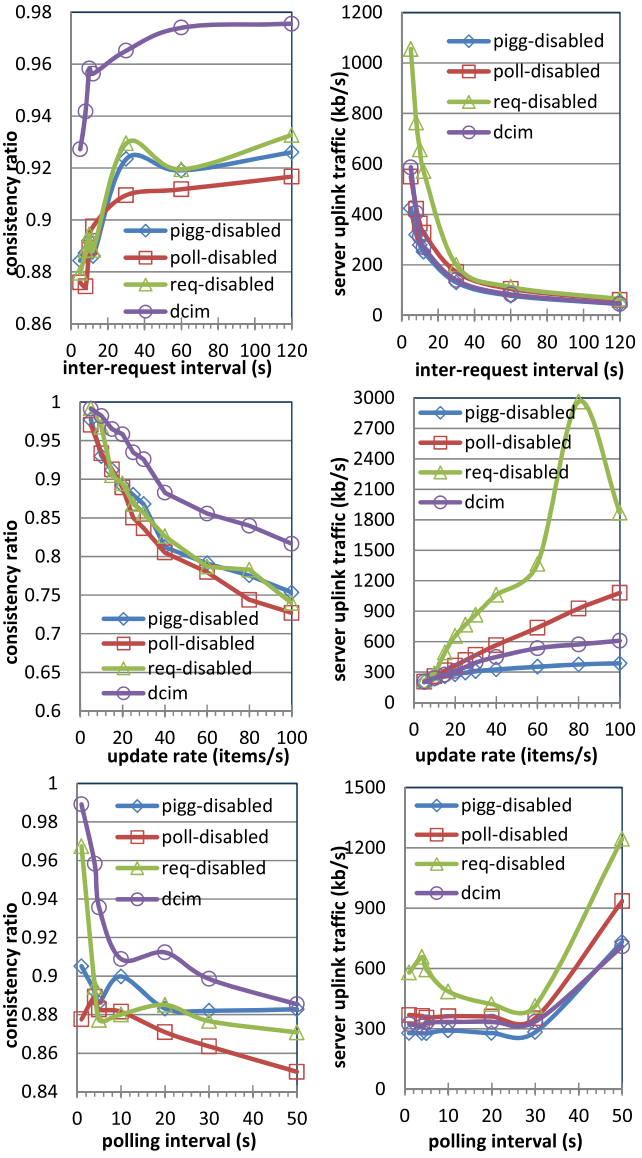

Fig. 12. Performance evaluation of DCIM mechanisms.

controls overhead traffic consumption. As for the update rate adaptation, its effect is evident from "poll-disabled" that has all the other features of DCIM disabled, but still shows higher consistency than fixed TTL.

5.8 Comparison with Push-Based Approaches

This section compares DCIM to SSUM [35] and to the updated invalidation (UIR) method [4]. In SSUM the server propagates item update information to the QDs, and computes for each cached item a ratio of its update rate to its request rate. If this ratio exceeds a threshold, the item is deleted from the server's state table, and no updates about it are propagated. However, if it falls below another threshold, the caching node receives updates for the item. Hence, SSUM reduces traffic associated with unnecessary updates for items that are more updated than requested. On the other hand, UIR is a server stateless approach that relies on invalidation updates broadcasted to the network to inform the nodes about the updated items. Caches answer the query after validating the requested data against the invalidation report. Unlike SSUM, UIR is not based on

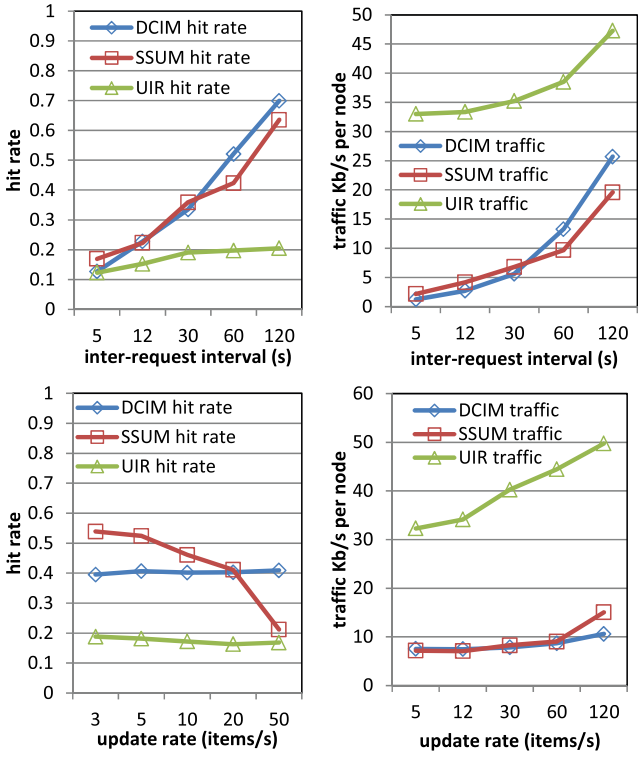

Fig. 13. Performance comparison between DCIM, SSUM, and UIR.

COACS. Hence, by also comparing these two methods, we additionally illustrate the advantage of COACS as an underlying caching system.

SSUM achieves consistency with a delta equal to the communication time between the server and caching nodes, but, as illustrated in Fig. 7 through Fig. 10, DCIM also provides consistency guarantees if the polling interval is not high, with a comparable generated traffic. We confirm the above analysis using three scenarios, with an area of $750 \times 750\ m^2$, a θ value of 0.5, a request period of 20 s, and an item size that was varied between 1 and 10 KB. All other parameters kept the same values as before.

We report the hit rate and node traffic versus the request interval and update rate. The θ value of 0.5 means there is more variety in the requested items, and given there is a total of 10,000 items, the probability of requesting an element several times is low, which reflects on the hit rate values in Fig. 13. For traffic, in SSUM it is due to maintaining the server state, and pushing data items proactively, while in DCIM it is due to validation requests and proactive fetching of items. UIR provides strong consistency but at the expense of more traffic and deteriorated data availability, while DCIM on the other hand provides near strong consistency as was shown earlier, but with considerably lower traffic and higher data availability. Moreover, Fig. 13 shows the effectiveness of COACS as a caching system in providing data availability while keeping traffic in the network low when compared to other caching systems. One can refer to [2] for detailed analysis of COACS and comparisons to existing systems.

We conclude the experimental results with Table 5, which summarizes key properties of DCIM and compares them to the presented pull-based approaches and SSUM.

TABLE 5
Comparison between DCIM and Other Approaches

Property	Poll Each Time	Fixed TTL
Type	Pull (client side)	Pull (client side)
Query Delay	High	Based on chosen TTL
Consistency	Highest	Depending on chosen TTL
Server Traffic	low	Low to medium
Practicality	No: high delays plus disconnections from server	No: TTL value does not work for all items in all scenarios
MANET traffic	Low	Low
Scalability	scalable	TTL value assignment not scalable
Property	**SSUM**	**DCIM**
Type	Push (server side)	Pull (client side)
Query Delay	Low	Low
Consistency	High	High, depending on polling interval
Server Traffic	Medium	Medium, < 10 Kb/s per node. (network bandwidth = 2 Mbps)
Practicality	Limited: requires server to maintain state of MANET	Most practical: it achieves low delay, high consistency ratio
MANET traffic	Medium	Medium
Scalability	Limited: Server keep s state of MANET	Scalable since it operates on the client side.

6 Conclusion

We presented a client-based cache consistency scheme for MANETs that relies on estimating the inter update intervals of data items to set their expiry time. It makes use of piggybacking and prefetching to increase the accuracy of its estimation to reduce both traffic and query delays. We compared this approach to two pull-based approaches (fixed TTL and client polling) and to two server-based approaches (SSUM and UIR). This showed that DCIM provides a better overall performance than the other client-based schemes and comparable performance to SSUM.

For future work, we will explore three directions to extend DCIM. First, we will investigate more sophisticated TTL algorithms to replace the running average formula. Second, we will extend our preliminary work in [44] to develop a complete replica allocation. Third, DCIM assumes that all nodes are well behaved, as issues related to security were not considered. However, given the possibility of network intrusions, we will explore integrating appropriate security measures into the system functions. These functions include the QD election procedure, QD traversal, QD and CN information integrity, and TTL monitoring and calculation. The first three can be mitigated through encryption and trust schemes [39], [40]. The last issue was not tackled before, except in the case of [41].

References

[1] T. Andrel and A. Yasinsac, "On Credibility of MANET Simulations," *IEEE Computer*, vol. 39, no. 7, pp. 48-54, July 2006.
[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] D. Barbara and T. Imielinski, "Sleepers and Workaholics: Caching Strategies for Mobile Environments," *Proc. ACM SIGMOD*, pp. 1-12, May 1994.
[4] G. Cao, "A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments," *IEEE Trans. Knowledge and Data Eng.*, vol. 15, no. 5, pp. 1251-1265, Sept./Oct. 2003.

[5] D. Li, P. Cao, and M. Dahlin, "WCIP: Web Cache Invalidation Protocol," IETF Internet draft, http://tools.ietf.org/html/draft-danli-wrec-wcip-01, Mar. 2001.
[6] J. Cao, Y. Zhang, G. Cao, and X. Li, "Data Consistency for Cooperative Caching in Mobile Environments," *Computer,* vol. 40, no. 4, pp. 60-66, 2007.
[7] P. Cao and C. Liu, "Maintaining Strong Cache Consistency in the World-Wide Web," *IEEE Trans. Computers,* vol. 47, no. 4, pp. 445-457, Apr. 1998.
[8] W. Li, E. Chan, D. Chen, and S. Lu, "Maintaining Probabilistic Consistency for Frequently Offline Devices in Mobile Ad Hoc Networks," *Proc. IEEE 29th Int'l Conf. Distributed Computing Systems,* pp. 215-222, 2009.
[9] J. Jung, A.W. Berger, and H. Balakrishnan, "Modeling TTL-Based Internet Caches," *Proc. IEEE INFOCOM,* Mar. 2003.
[10] B. Krishnamurthy and C. Wills, "Study of Piggyback Cache Validation for Proxy Caches in the World Wide Web," *Proc. USENIX Symp. Internet Technologies and Systems,* Dec. 1997.
[11] J. Lee, K. Whang, B. Lee, and J. Chang, "An Update-Risk Based Approach to TTL Estimation in Web Caching," *Proc. Third Int'l Conf. Web Information Systems Eng. (WISE '02),* pp. 21-29, 2002.
[12] D. Wessels, *Squid Internet Object Cache,* http://squid.nlanr.net, Aug. 1998.
[13] Y. Huang, J. Cao, Z. Wang, B. Jin, and Y. Feng, "Achieving Flexible Cache Consistency for Pervasive Internet Access," *Proc. IEEE Fifth Ann. Int'l Conf. Pervasive Computing and Comm.,* pp. 239-250, 2007.
[14] O. Bahat and A. Makowski, "Measuring Consistency in TTL-Based Caches," *Performance Evaluation,* vol. 62, pp. 439-455, 2005.
[15] M. Denko and J. Tian, "Cooperative Caching with Adaptive Prefetching in Mobile Ad Hoc Networks," *Proc. IEEE Int'l Conf. Wireless and Mobile Computing, Networking and Comm. (WiMob '06),* pp. 38-44, June 2006.
[16] J. Jing, A. Elmagarmid, A. Helal, and R. Alonso, "Bit-Sequences: An Adaptive Cache Invalidation Method in Mobile Client/Server Environments," *Mobile Networks and Applications,* vol. 2, pp. 115-127, 1997.
[17] Q. Hu and D. Lee, "Cache Algorithms Based on Adaptive Invalidation Reports for Mobile Environments," *Cluster Computing,* vol. 1, pp. 39-50, 1998.
[18] Z. Wang, S. Das, H. Che, and M. Kumar, "A Scalable Asynchronous Cache Consistency Scheme (SACCS) for Mobile Environments," *IEEE Trans. Parallel and Distributed Systems,* vol. 15, no. 11, pp. 983-995, Nov. 2004.
[19] S. Lim, W.C. Lee, G. Cao, and C. Das, "Cache Invalidation Strategies for Internet-Based Mobile Ad Hoc Networks," *Computer Comm.,* vol. 30, pp. 1854-1869, 2007.
[20] K.S. Khurana, S. Gupta, and P. Srimani, "A Scheme to Manage Cache Consistency in a Distributed Mobile Wireless Environment," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 7, pp. 686-700, 2001.
[21] V. Cate, "Alex - A Global Filesystem," *Proc. USENIX File System Workshop,* pp. 1-12, May 1992.
[22] L. Yin and G. Cao, "Supporting Cooperative Caching in Ad Hoc Networks," *IEEE Trans. Mobile Computing,* vol. 5, no. 1, pp. 77-89, Jan. 2006.
[23] G. Cao, L. Yin, and C. Das, "Cooperative Cache-Based Data Access in Ad Hoc Networks," *Computer,* vol. 37, no. 2, pp. 32-39, 2004.
[24] X. Tang, J. Xu, and W-C. Lee, "Analysis of TTL-Based Consistency in Unstructured Peer-to-Peer Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 19, no. 12, pp. 1683-1694, Dec. 2008.
[25] L. Bright, A. Gal, and L. Raschid, "Adaptive Pull-Based Policies for Wide Area Data Delivery," *ACM Trans. Database Systems,* vol. 31, no. 2, pp. 631-671, 2006.
[26] C.P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell, "A Hierarchical Internet Object Cache," *Proc. Ann. Conf. USENIX Ann. Technical Conf.,* p. 13, 1996.
[27] V. Jacobson, "Congestion Avoidance and Control," *Proc. ACM SIGCOMM Computer Comm. Rev.,* vol. 25, p. 187, 1995.
[28] X. Chen and P. Mohapatra, "Lifetime Behavior and its Impact on Web Caching," *Proc. IEEE Workshop Internet Applications,* pp. 54-61, 1999.
[29] Y. Sit, F. Lau, and C-L. Wang, "On the Cooperation of Web Clients and Proxy Caches," *Proc. 11th Int'l Conf. Parallel and Distributed Systems,* pp. 264- 270, July 2005.
[30] U.A. Ninan, M. Raunak, P. Shenoy, and K. Ramamritham, "Maintaining Mutual Consistency for Cached Web Objects," *Proc. 21st Int'l Conf. Distributed Computing Systems,* p. 371, 2001.
[31] N. Chand, R. Joshi, and M. Misra, "A Zone Co-Operation Approach for Efficient Caching in Mobile Ad Hoc Networks," *Int'l J. Comm. Systems,* vol. 19, pp. 1009-1028, 2006.
[32] Y. Du and S.K.S. Gupta, "COOP - A Cooperative Caching Service in MANETs," *Proc. Joint Int'l Conf. Autonomic and Autonomous Systems and Int'l Conf. Networking and Services (ICAS-ICNS),* pp. 58-58, Oct. 2005.
[33] Y. Du, S. Gupta, and G. Varsamopoulos, "Improving On-Demand Data Access Efficiency in MANETs with Cooperative Caching," *Ad Hoc Networks,* vol. 7, pp. 579-598, 2009.
[34] C.C. Holt, "Forecasting Seasonals and Trends by Exponentially Weighted Moving Averages," *Int'l J. Forecasting,* vol. 20, no. 1, pp. 5-10, 2004.
[35] K. Mershad and H. Artail, "SSUM: Smart Server Update Mechanism for Maintaining Cache Consistency in Mobile Environments," *IEEE Trans. Mobile Computing,* vol. 9, no. 6, pp. 778-795, June 2010.
[36] B. Krishnamurthy and C.E. Wills, "Piggyback Server Invalidation for Proxy Cache Coherency," *Proc. Seventh Int'l Conf. World Wide Web,* Apr. 1998.
[37] Y. Fang, Z. Haas, B. Liang, and Y.B. Lin, "TTL Prediction Schemes and the Effects of Inter-Update Time Distribution on Wireless Data Access," *Wireless Networks,* vol. 10, pp. 607-619, 2004.
[38] G. Zipf, *Human Behavior and the Principle of Least Effort.* Addison-Wesley, 1949.
[39] N. Boudriga and M. Obaidat, "Fault and Intrusion Tolerance in Wireless Ad Hoc Networks," *Proc. IEEE Wireless Comm. Networking Conf.,* 2005.
[40] P. Papadimitratos and Z. Haas, "Secure Data Transmission in Mobile Ad Hoc Networks," *Proc. ACM Workshop Wireless Security,* pp. 41-50, 2003.
[41] W. Zhang and G. Cao, "Defending Against Cache Consistency Attacks in Wireless Ad Hoc Networks," *Ad Hoc Networks,* vol. 6, pp. 363-379, 2008.
[42] H. Maalouf and M. Gurcan, "Minimisation of the Update Response Time in a Distributed Database System," *Performance Evaluation,* vol. 50, no. 4, pp. 245-66, 2002.
[43] T. Hara and S. Madria, "Dynamic Data Replication using Aperiodic Updates in Mobile Ad Hoc Networks," *Proc. Database Systems for Advanced Applications,* pp. 111-136, 2004.
[44] K. Fawaz and H. Artail, "A Two-Layer Cache Replication Scheme for Dense Mobile Ad Hoc Networks," *Proc. IEEE Global Comm. Conf. (GlobeCom),* Dec. 2012.

Kassem Fawaz received the BE degree with high distinction and the ME degree in computer and communications engineering from the American University of Beirut in 2009 and 2011, respectively. Currently, he is a PhD candidate at the University of Michigan, where he is doing work in the areas of mobile computing and privacy. He received the Distinguished Graduate Award upon graduation in 2009, and has published 15 papers in web systems and pervasive computing. He is a student member of the IEEE.

Hassan Artail received the BS degree with high distinction and the MS degree in electrical engineering from the University of Detroit in 1985 and 1986, respectively, and the PhD degree in electrical and computer engineering from Wayne State University in 1999. He is a professor at the American University of Beirut (AUB), where he is doing research in Internet and mobile computing. During the past 10 years, he has published more than 150 papers in top conferences and reputable journals. Before joining AUB in 2001, he was a system development supervisor at the Scientific Labs of Daimler-Chrysler, where he worked for 11 years with system development for vehicle testing applications. He is a senior member of the IEEE.