An Effective Management Model for Data Caching in MANET Environment

Amer Abu Salem
College of Information Technology, Zarqa University, Jordan
abusalem@zu.edu.jo

Abstract: *A mobile Ad-hoc (MANET) network has the main challenge to provide the needed data for the desired mobile nodes. An efficient on request routing protocol for MANET is Ad-hoc on-demand Distance Vector (AODV), which is based on two main methods: route discovery and route maintenance. Route discovery is the process used to detect a route to the destination from the packet source, while route maintenance is the process used to detect a link failure and repair it. Cooperative caching tends improving data availability in mobile ad-hoc networks, the coordination of cache discovery and cache management strategies is very significant in the cooperative caching of MANETs because requests for data and answers to requested data can be reduced simply due to interference, network congestion, or when a forwarding node is out of reach and the route breaks down. Cooperative cache management is much more complicated in cooperative caching because it also depends on neighbouring nodes to decide what to cache. In this paper, three algorithms were proposed: (1) a combination algorithm for cache admission control based on cache data and location of data to save space and reduce data redundancy, (2) a value-based policy for cache placement and replacement instead of the more common least recently used strategy, depending on metrics that describe cached items to increase the local cache hit ratio, and (3) a combined algorithm for cache consistency that includes time-to-live, pull, and push policies to enhance data availability and system scalability. The proposed algorithm implemented by the NS3 simulation program; which used to create a network using the AODV protocol in several parameters and achieve better system performance.*

Keywords: *MANET, NS3, admission control, replacement strategy, consistency strategy.*

Received May 15, 2023; accepted September 25, 2023
https://doi.org/10.34028/iajit/20/6/1

1. Introduction

A Mobile Ad-hoc Network (MANET) is a type of wireless network in which mobile devices, also called nodes, communicate with each other without the need for a central access point or infrastructure. MANETs are widely used in military and civilian applications where communication infrastructure is limited, unreliable or unavailable, such as disaster relief and battlefield communication systems. MANETs have several key characteristics that differentiate them from other types of networks. First, they are self-organizing, which means that nodes communicate with each other and dynamically establish communication paths without the need for a pre-existing infrastructure. Second, they are highly dynamic, which means that nodes can move freely and unpredictably, making the network topology constantly changing. Third, they are highly decentralized, which means that there is no central authority controlling the network, and each node has equal importance in the network [13].

MANETs have several advantages over other types of networks. First, they are highly flexible, allowing communication in areas where other types of networks cannot function, such as in remote areas or in disaster scenarios. Second, they are highly scalable, which means that they can easily adapt to changes in the number of nodes in the network. Third, they are highly reliable, as nodes can communicate with each other through multiple paths, reducing the chances of network failure. Despite their advantages, MANETs also face several challenges. First, they are vulnerable to security threats, as the lack of a central authority makes it difficult to secure the network. Second, they face challenges in routing, as the highly dynamic nature of the network requires the use of complex routing protocols to ensure efficient communication. Third, they face challenges in quality of service, as the network may experience congestion due to limited bandwidth and the unpredictable movement of nodes [12].

Data caching is a technology which improves the availability of data and access to any network information. Wireless ad-hoc networks present a number of special challenges to the issue of cooperative caching compared to infrastructure-based networks. In MANETs, the nodes always move, enter and leave the network and therefore change the topology of the network constantly. Mobile nodes are fully decentralized with no single controlling entities; nodes cannot communicate directly with each other usually, have limited battery power, operate with the distance between nodes, consume bandwidth and therefore cannot to do complex tasks. These constraints and other issues are motivation to study all the functions and subprocedures in the development of a cooperative caching strategy, which enhances data availability and access efficiency in MANET.

In the ad-hoc network, multi-hop wireless links connect each other to the nodes, and each mobile node plays as a router, receiving and forwarding data for the others; this is a consequence of a deficiency of infrastructure support. In many of the previous research in ad-hoc networks, dynamic protocols have been improved, allowing routes between two communication nodes to be effectively detected. Although it is mainly routing, the ad-hoc network's primary objective is to support data-based mobile nodes. The cache is a small, intelligent memory device that temporarily stores data. The majority of the MANET data exchanged is dynamic. Therefore the small cache space that is available must be used efficiently. Almost all-dynamic information is time-critical and after a certain period of time cannot be used. This time we're talking about an impasse, any data can be removed from the caching if the impasse expires to make cache space easier to use. The data server is a highly efficient data system that keeps and manages data from mission-critical transactional applications in real-time [6].

Cooperative caching in a MANET is a very significant strategy, which allows the sharing and coordination of cached data between mobile nodes for future data server retrieval. When many mobile nodes are seeking the same data on the same server, heavy traffic near the data server is inevitable. This high load is reduced on the data server with the aim of cooperative caching [10]. This reduces network resource efficiency such as bandwidth and power as well as reducing data recovery from a remote data server. Many previous types of research used this strategy to improve web performance in wired networks. On the other side, resource constraints and node mobility have limited the application of this technique in ad-hoc networks.

The hybrid caching strategy is a combination of other caching strategies and schemes. It faces two important challenges in cooperative caching: firstly, how to efficiently locate a cache within the entire cooperative cache system that contains the desired data item (known as cache discovery), and secondly, how to manage the local cache to enhance the capabilities of the collaborative caches (known as cache management). Both these challenges encompass several issues, procedures, and functions that work together to produce significant improvements in terms of request success ratio, cache hit ratio, and average query latency compared to other caching strategies.

This study recommends making some modifications and improvements to the existing algorithm to create a new strategy model that can enhance data accessibility and minimize the local cache miss ratio.

The following paper was organized as follows: Section 2 provides an outline of the cache admission control strategy, followed by section 3, which details the cache placement and replacement strategy, including a new model. Section 4 proposes a hybrid consistency strategy that utilizes the Time-to-live algorithm in combination with Pull and Push mechanisms. In section 5, the scenario simulation used to evaluate the effectiveness of the proposed strategies in AODV is described. Section 6 summarizes the results obtained from the NS3 simulator and the case scenario evaluation. Lastly, section 7 presents the conclusions of the study.

2. Cache Admission Control Strategy

When a node receives the requested data, the cache admission control is used to determine whether a data item should be placed in the cache. The addition of a data item to the cache may not always be positive because the probability of cache hits may be reduced by wrong judgment. For example, to add a data item for a local cache which is limited storage leads to replacement strategy, and replacing a data item accessible sooner with an item which is not accessed soon will reduce network performance [9].

2.1. Hybrid Cache Admission Control Approach

In this paper, the hybrid cache admission control algorithm allows a node to cache the data item or location of data based on different criteria [15]. The following criteria are:

- The distance between the requester and the source of the requested data item, measured in terms of the number of intermediate nodes or hops traversed, is known as the hop count. (Δ hops).
- The size of a data item, denoted as (Sx), refers to the amount of memory or storage space required to store that particular data item. The size of a data item is an important factor to consider in caching strategies, as it affects the amount of storage space required for caching and the efficiency of data retrieval.
- Time-to-live refers to the maximum amount of time that a packet is allowed to remain in a network before it is discarded. It is a field in the packet header that is decremented by one at each hop and the packet is discarded when the value reaches zero. This mechanism is used to prevent packets from circulating indefinitely within the network and to free up network resources (TTLx).

The following heuristics are employed to determine whether to cache or locate a particular data item:

- If the distance between the requesting node and the data server is Δ hops or less, then it is recommended to follow the cache location strategy to conserve cache space within the same cluster. This approach can improve the cache hit ratio and reduce the average query latency by minimizing the need to search for data items in remote caches.

- Otherwise, if there is enough available space in the cache, it is recommended to cache the data to enhance system performance. However, if the requested data item is already available within Δ hops from the requesting node, caching may be unnecessary to avoid duplication of the same data item in the same cluster. This is because the cached data can be easily accessed by the nodes located closely. Therefore, during simulation experiments, this study replicated the same data item at least two hops away to avoid unnecessary duplication within the same cluster.
- If S_x is small, it is advisable to cache the data since only a very small part of the cache is required for the data item; otherwise, the cache location should be used to save space. The threshold value for the data size is T_S.
- If TTLx is small, the location of the cache is not the best choice, as the data item may soon be invalid. Using cache the location it may cause the wrong path to be chased and resend the request to the data server. In this situation, cache the data should be used. If TTL_X is large, you should cache the location. The TTL threshold is a system parameter and is referred to as T_{ttl}.

2.2. Hybrid Packet Sniffing Approach

Wireless networks are characterized by the fact that any device within range can intercept packets sent by others. However, nodes that are not interested in the packets will discard them. While this behavior can be problematic for resource discovery in dynamic networks, it can also be useful for caching certain resources. For instance, intermediate nodes can cache resource locations to help reduce the time required for resource discovery.

In a dynamic network where wireless nodes frequently move around, resource discovery can be challenging, leading to long delays or even looping. By caching resource locations, this issue can be alleviated. The proposed approach in the referenced paper leverages the packet-sniffing capability of wireless networks to intercept reply return messages and cache resource locations. This technique is particularly effective in ad-hoc networks, where all nodes are expected to be active most of the time in order to participate in the network. As a result, sniffing passing packets does not lead to additional power consumption [4].

This paper applied a hybrid caching method in data item locations called hit/miss caching strategy. This caching method basically means using a Pre-request table to cache the location for both source and requester when a data reply is intercepted or cache the location of a requester when a cache miss to predict possible location if there is no cache requested data item. If a cache miss entry is found with the possible sources, it means that some time ago the same data item was requested, and can predict if that data item could have been transferred to the node requester. Now instead of forwarding the request to Data-Clusterhead or next hop, it may only send the request in the direction of that node [14]. Figure 1 illustrates the proposed cache admission control policy.

```
Notation:
$d_x$ : ID of data item.
$D_x$ : The content of the data item $d_x$ .
$MN_i$ : Mobile node number i .
$MN_i^1$ : Set of one-hop neighbours of node $MN_i$ .
$S_x$ : Size of data item $d_x$ .
$T_s$ : Threshold value for data size.
$TTL_x$ : time-to-live of $d_x$
$T_{TTL}$ : the threshold value TTL .

Pseudo code:
Admissin_control ($d_x$,$MN_i$)
 Begin
   When the data item $D_x$ is obtained from $MN_j$
     If ($d_x$ is requested by the current node $MN_i$)   then
       If ($MN_j \notin MN_i^1$)   then             // if $MN_j$ is not one-hop neighbour of $MN_i$
         If ($S_x < T_s$ or $TTL_x < T_{TTL}$)   then        //  if $D_x$ is a small size or has a small TTL
               If there sufficient space to cache the item $D_x$  then
                     Cache data item $D_x$ and the source location $MN_j$
               Else Replacement_policy ($D_x$)      // there is no free space for new item
       Else
               Cache the source location $MN_j$
       Else
       Cache the source location $MN_j$             // if $MN_i$ is one-hop neighbour of $MN_i$
    Else
   If ($MN_k \notin MN_i^2$ and $MN_k \notin MN_j^2$) then        // if current node MNk is not two-hop neighbour of source and
                                                  requester
       If ($S_x < T_s$ or $TTL_x < T_{TTL}$)   then
               If there sufficient space to cache the item $D_x$  then
                 Cache data item $D_x$ and the location $MN_j$ and $MN_i$
               Else
                 Cache the source and destination locations $MN_j$ and $MN_i$
 End
```

Figure 1. An algorithm of the proposed cache admission control.

2.3. An Illustrative Example

As shown in Figure 2, assume MN_1 sends a request for a data item, the UK map, and MN_2, MN_3, MN_4 are located along the route through which the request reaches the MN_5 server. Node MN_1 requests the UK map and sends a lookup message to its Data-Clusterhead then to the next hop in the path to the data server, cache miss status will be stored in Pre-Request cache table in MN_2, MN_3, MN_4 and their Data-Clusterheads.

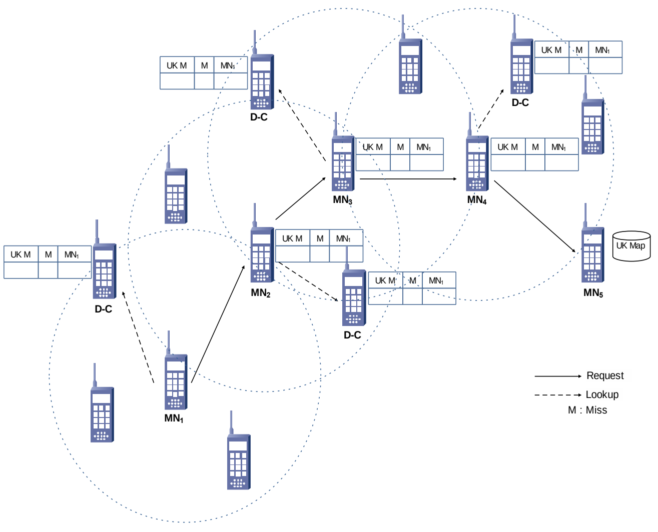


Figure 2. Example of hybrid cache admission control, case 1.

As shown in Figure 3, when node MN_5 sends a reply message with an identifier of location and request node ID, nodes MN_2, MN_3, MN_4 change the status of the request from miss to hit in Pre-Request cache table and send update message to their Data-Clusterhead.

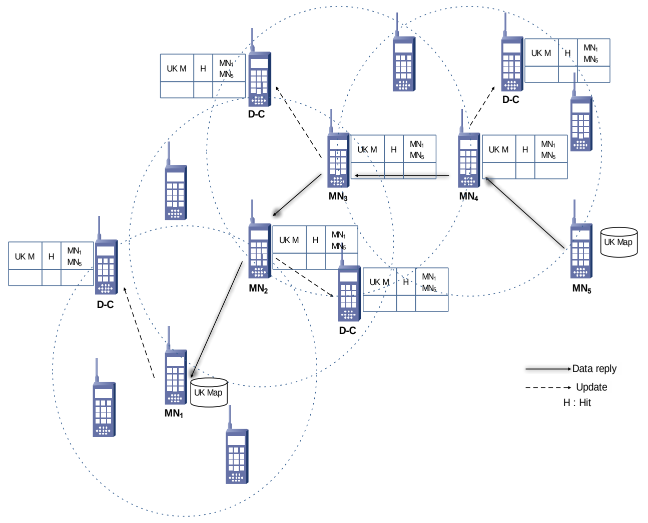


Figure 3. Example of hybrid cache admission control, case 2.

In Figure 4, node MN_4 has been moved far away from MN_5 so the connection was broken, so a new routing path is initialized. MN_5 will reply message to MN_6 then MN_6 will cache the data ID and its locations MN_1 and MN_5 in the Pre-Request table and forward back to requester node MN_1. At this moment cache miss in Pre-Request on MN_3, MN_4 and their D-Clusterheads will be useful to predict the location of the UK map as a data item in the near future since MN_1 has the UK map.

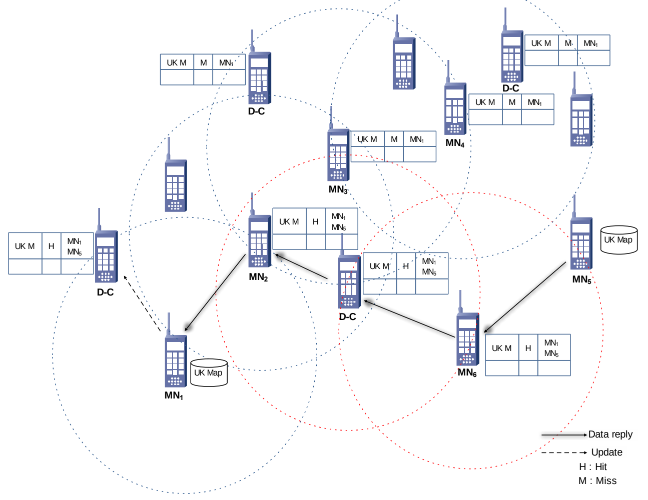


Figure 4. Example of hybrid cache admission control, case 3.

Next, Figure 5 illustrates the node MN_3 cached the data item, the UK map, while it data packet sniffing and after it applied the cache admission control algorithm that is used in overall strategy, because the data source is not one hop neighbour away and the data item is small enough in size that is available sufficient space to cache the data itself.

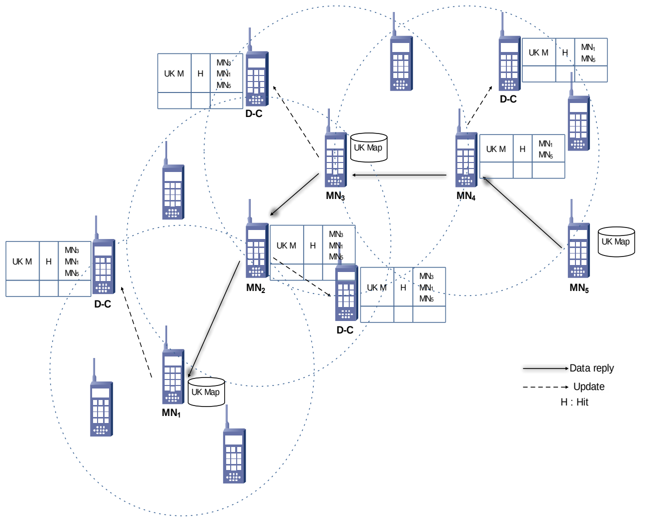


Figure 5. Example of hybrid cache admission control, case 4.

3. Hybrid Cache Placement and Replacement Approach

A cache replacement strategy is required if a mobile node needs to cache a data item, but the local cache is full. Consequently, a suitable subset of data items must be found to be removed from the cache. In operating systems, management of virtual memory and management of database buffer, cache replacement policy has been studied extensively. However, these algorithms may not be suitable for ad-hoc networks for several reasons:

- In ad-hoc situations, the data item size cannot be fixed; the replacement strategy used must handle data items of various sizes.
- The transfer time of the data item depends on the item size and the number of hops between the requested node and the data source or server. Therefore, the cache hit ratio may not be the most correct quality measurement for a cache replacement strategy.
- The replacement strategy should also take into account cache consistency. In other words, data items that are not consistent earlier must be replaced earlier.

This paper presents how and where to place the data item in cluster members when a node receives a data item from the source and decides to cache it if there is no enough space in its local cache, as follows:

- When a node MNi receives a data item and decides to cache it, the node caches the data item if the local cache space is sufficient after it has removed all invalid data.
- Otherwise, MNi sends the D-Clusterhead message to ask it to check its cluster members' cache space available. If any cluster member's available cache space is enough to store the data item, the MNi node transmits the data.

- If the cache space available for each cluster member is not sufficient to cache the received data element, MNi Executes a value-based cache replacement strategy, where data items with the smallest value are those removed from the local cache. To implement a value-based cache replacement strategy, four factors are taken into account when calculating the value of a data item at a node [11].
- Popularity: this factor indicates how frequently nodes access a particular data item. To replace an item in the cache, the item with the lowest probability of access is selected. To determine the probability of access (Px) for each data item (dx), a node maintains a PreReq table. Initially, Px for each item is set to zero. As nodes request items, the corresponding Px values are incremented accordingly. This allows the node to track the popularity of each data item over time and make informed decisions about cache replacement.
- Distance: this factor is based on the number of intermediate nodes between the requested node and the server or data source that provides the data item. Distance is integrated into the cache replacement selection process as an important factor. When considering distance in cache replacement, data items that save bandwidth by being cached further away and reduce latency for later requests are given higher value. Therefore, if two data items have similar popularity, the item that is farther away but still within a reasonable distance may be selected for replacement over a closer but less useful item.
- Coherency: it is determined using the Time-To-Live (TTL) field for each data item, which specifies the period for which the item remains valid. For cache replacement, an item with a shorter TTL should be selected as it is closer to expiration. This factor ensures that cached data items remain coherent with the original source, preventing stale or outdated data from being served. When an item's TTL has expired, the cache should discard it and fetch a fresh copy from the original source. By selecting data items with shorter TTLs for replacement, the cache can maintain data coherency and reduce the probability of serving outdated data.
- Size: for replacement, a data item with the larger size S_x should be preferred as the cache may contain additional data items and respond to further data requests.

On the basis of the above about residual access popularity, distance, coherency and data size, it is obvious that a data item dx is the appropriate data item subset to remove from the cache if its P_x is the lowest, its D_x is the lowest, its $TTL_{xremain}$ is the smallest, and its S_x is the largest. In other words, a node with the smallest weight is the best selection of data item to remove when these four factors combine together as the weight, these metrics have different units, as:

- The popularity can theoretically vary between one and number of nodes *N*, a normalized translation is needed. One way to do it is:

$$P \rightarrow \frac{P}{N} \tag{1}$$

- The distance can theoretically vary between 1 and *N-1*, a normalized translation is needed. One way to do it is:

$$D \rightarrow \frac{D}{N-1} \tag{2}$$

- The theoretical coherence can vary between 0 and *TTL* 's initial value is set to δ, a normalized translation is needed. One way to do it is:

$$TTL_{remain} \rightarrow \frac{TTL_{remain}}{\delta} \tag{3}$$

- The data size can theoretically vary between 1 and maximum data size Smax, a normalized translation is needed. One simple way to do it is:

$$S \rightarrow \frac{S}{S_{max}} \tag{4}$$

Using the result from the above, the combined weight *Wx* for each cache data item *x* is,

$$HyCC\_repval = w1 \cdot \frac{P_x}{N} + w2 \cdot \frac{D_x}{N-1} + w3 \cdot \frac{TTL_{remain}}{\delta} + w4/\frac{S}{S_{max}} \tag{5}$$

Where *w1*, *w2*, *w3*, *w4* are weight factors such that $\sum$j=1 w_j=1 and $0 \leq w_j \leq 1$. And *x* is the ID number of the cache data item. For replacement, a data item with the lowest value of this function is considered. The nodal block diagram as Figure 6 is based on the above description. The decisions of each node are based on the cache information from the cache state table in the local cache.

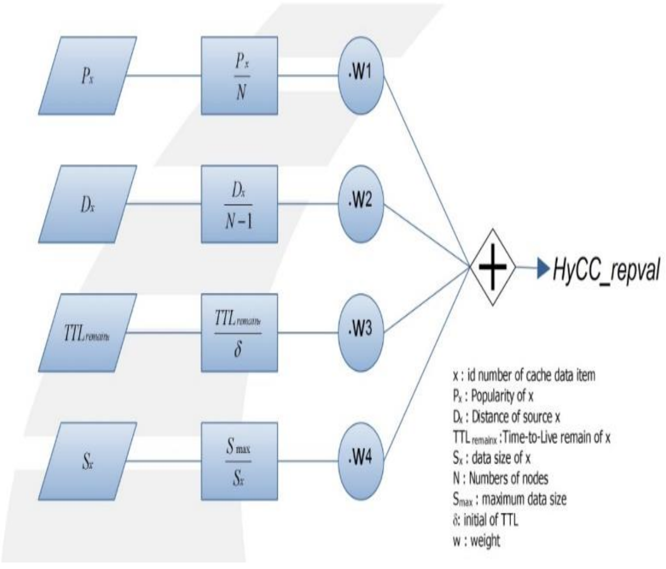
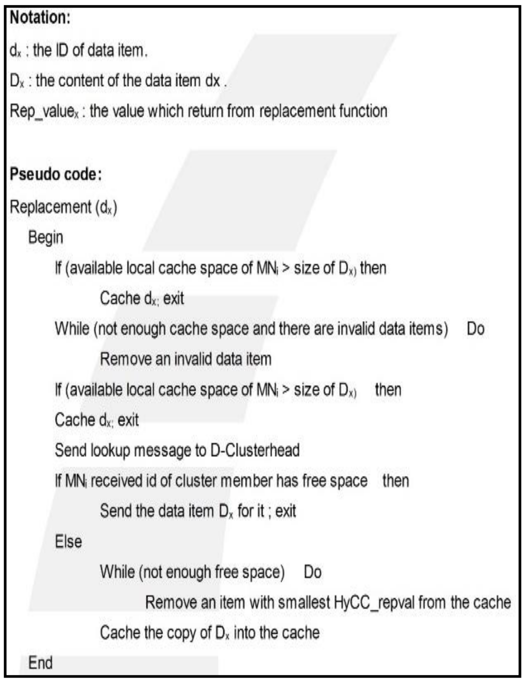


Figure 6. Block diagram for the cache manager.

The proposed cache placement and replacement algorithm is demonstrated in Figure 7.



Notation:

d_x : the ID of data item.

D_x : the content of the data item dx .

Rep_value$_x$: the value which return from replacement function

Pseudo code:

```
Replacement ($d_x$)
  Begin
      If (available local cache space of $MN_i$ > size of $D_{x)}$ then
            Cache $d_{x;}$ exit
      While (not enough cache space and there are invalid data items)    Do
            Remove an invalid data item
      If (available local cache space of $MN_i$ > size of $D_{x)}$    then
      Cache $d_{x;}$ exit
      Send lookup message to D-Clusterhead
      If $MN_i$ received id of cluster member has free space   then
            Send the data item $D_x$ for it ; exit
      Else
            While (not enough free space)    Do
                  Remove an item with smallest HyCC_repval from the cache
            Cache the copy of $D_x$ into the cache
  End
```

Figure 7. The proposed cache placement and replacement strategy.

4. Cache Consistency Strategy

Cooperative caching is a very significant strategy; this covers the sharing and coordination between multiple nodes of the cached data and may be used for improved data availability and system scalability, as well as to reduce demand for wireless bandwidth and mobile node battery power. One of the critical problems in cooperative caching is how to maintain caching consistency, namely the coherence between the original data of the server node and the replicated data received from the caching nodes.

The consistency strategy of the cache should be covered to certify that nodes only access valid data states. In many other systems, such as multi-processor architectures, distributed database systems, distributed shared memory and client-server systems, cache consistency problems have been investigated. Two consistency models have generally been used: The weak consistency and the strong consistency model [7]. In the weak consistency model, invalid data can be sent to the node. In a strong consistency model, no old copy of the modified data item will be sent to the node after an update is completed.

One of the goals of developing a cooperative caching scheme is to preserve the determined level of consistency at the minimum possible cost. Algorithms for the maintenance of cache consistency can be categorized into two primary classes: Stateless and stateful on the basis of maintaining the cache status in the data node [5].

Existing MANET consistency maintenance processes are generally stateless, where the node of the data server does not know the cached data status. The server node basically depends on the transmission mechanism for the diffusion of data updates which will necessarily generate a great deal of redundant data updates.

The data server node keeps cache status (TTL value) of each cached copy in stateful constancy maintenance algorithms. Therefore, the data server node can transmit data updates selectively to the cache nodes that need updates on the basis of the maintained cache status. In other words, cache nodes with a cached copy presently expire. In comparison with stateless algorithms, stately algorithms importantly reduce maintenance costs of consistency by selectively broadcasting cache-based data updates. The stateful procedures are therefore more appropriate for MANETs because data transmission power consumption for the broadcast of data updates is considerably higher than the power consumption of wireless node local cache status maintenance calculations [8].

In this section, this paper proposed a stateful cache consistency maintenance approach to achieving a weak consistency in cooperative caching in MANETs. Where the cache query rates and the time to live value for every cache are maintained in the data server node.

Depending on cache status, the data server can choose which cache copies to broadcast or will soon be invalid so all data updates are therefore required for each update. If data update receivers have been determined, the proposed cache consistency strategy applies a specific approach to broadcast the data update among the selected caching nodes, which gives up consistency in order to reduce the cost and query delay. Advance nodes also refresh a cached data item and its TTL if a fresh copy of the same data is passed through using a sniffing mechanism [1].

Also, cache consistency in both data caching and data caching due to the expiry of TTL; certain cached data or location may be invalidated. In general, the cache removes invalid data. Invalid data can be helpful sometimes. Since the node cached this data, this shows that the node is interested in these data. If a node transmits a data element and finds that an invalid copy of that data is in the cache, the data are updated to be used in future. In the proposed model, to save space, when the cached data item expires, it is removed from the cache while its ID is maintained as an invalid state as an indication of the value of the node in the PreReq table. Surely, the interest of a node may change, and the expired position of data should not be kept in a table forever. If the location of the expired database item has not been replaced during the duration of TTL, it is removed from the table.

The proposed cache consistency algorithm used in the model assumes a commonly used system model. Each data item is connected to a single node it is able to modify the source data, and that node is mentioned as the node of the data server. A group of nodes called caching nodes can cache each data item. The caching nodes hold copies of the data items are named cache copies. Two fundamental mechanisms of cache consistency are provided: push procedure and pull procedure. The server node pushes the cache nodes to update the data. The cache node sends a query to the data server using pull to assure whether the cache copies are up-to-date or not.

We proposed a hybrid consistency strategy based on Time to live algorithm with Pull and Push mechanisms [2] as follow:

- Time-To-Live (TTL): each cached copy is matched to one timeout value TTL in this mechanism. The initial value of TTL is set to δ. The caching node can directly reply to a requested data from its local cache when TTL is unexpired that is when TTL greater than 0.
- Pull-Strategy: when the application requests a data item which is invalid, First, the cache node pulls the server in order to the cache copy updating and renew the TTL to δ. After that, the cache node is able to respond immediately to the requesting node with the requested data from its local cache. This approach ensures that the difference between the cached copy and data from the source is not exceeded over δ by matching a TTL value with each cached copy. Although the pull guarantee with the TTL algorithm is not cost-efficient usually, it is due to the pull mechanism's round trip maintenance costs. This investigation uses a stable selective push mechanism to economize on maintaining consistency.
- Push-Strategy: the server node changes the caching nodes for the data update with the push mechanism, it only entails unidirectional maintenance costs (traffic overhead, query latency, etc.,). It is therefore effective to use the principles of the design of the push mechanism to reduce the cost of maintenance of consistency, but should only be pushed if it is expected that the cache will serve the requests and push if no other update is probably available. With regard to the cache status of a cache node, as shown in Figure 8 the original data update at t_u and through push or pull mechanism the TTL of this cached copy was refreshed at time t0. Hence, the remaining TTL of this caching node at time t_u is $TTL_{remain} = t_0 + \delta - t_u$ and the TTL refreshed is $TTL_{refresh} = tu - t0$.

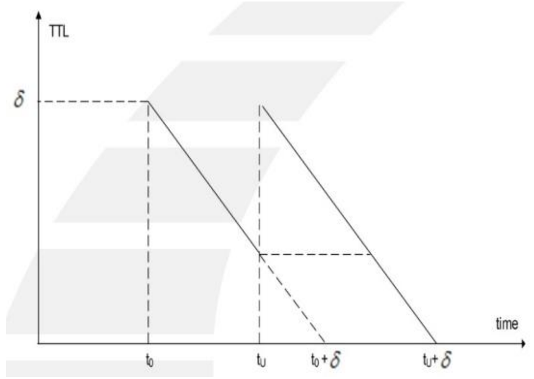


Figure 8. Cache status on one caching node.

The procedure of the cache consistency algorithm depends on the following functions:
The data server node detects the caching nodes to be updated in one push set.

1. The data server node determines how data updates are broadcast among the cache nodes selected.
2. Broadcast the push message containing the update of the data item and the push- set members ' IDs to all push- set caching nodes.
3. The data server unicasts the push message to the push- set's closest caching node.
4. The caching node receiving the push message removes its identification from the push- set.
5. The receiver node answers the sender with a message of acknowledgement.
6. The receiver node resends the push message to the closest caching node in the push-set.
7. This procedure is repeated until the push-set is empty.
8. The last caching node sends a message of acknowledgement to the data server node initiating the push message broadcast procedure.
9. If a recognition message is not received by the sender of a push message, it will wait t seconds. The push message will then be sent to the other in the push- set or the acknowledgement message will be sent to the data server node.
10. Once the data server node receives the acknowledgement message, will detect the time duration of the push message broadcast process. The caching nodes that the push message has received are also acknowledged.
11. The data server node determines the TTL values of all cache copies that receive the push message as δ - the duration of the push message broadcast process. The data server node can, therefore, more perfectly maintain the TTL values of each cached copy, considering the time delay in the broadcasting transmission of the push message.

- On the data server node

<u>Upon each data update source</u>

1. Create the push-set by adding the caching nodes into the push set.
2. Send a push message with the source data update and the push-set information to the closest caching node in the push-set.

<u>Upon each push message receiver</u>

3. Determine all caching nodes receiving the push message and the push process duration.
4. Update the TTL of the nodes receiving the push message by setting TTL= δ – duration time of push message process.

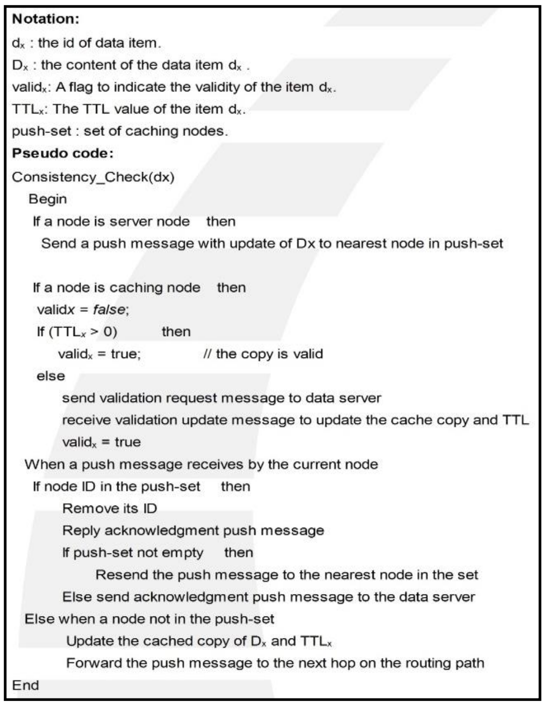
- On a caching node

<u>Upon cache request receiver</u>

1. If (TTL=0) update the cache copy and TTL from the data source node.
2. Process the cache request.

Upon push message receiver

3. Reply with acknowledgement of the push message.
4. Remove its ID from the push-set.
5. If (push-set size > 0) resend the push message to the push-set's closest caching node.
6. Else, send the acknowledgement message to the data server node.
7. Send the acknowledgement message to the data server node, if no push message is received within the time period.



Notation:

d_x : the id of data item.
D_x : the content of the data item d_x .
$valid_x$: A flag to indicate the validity of the item d_x.
TTL_x: The TTL value of the item d_x.
push-set : set of caching nodes.

Pseudo code:

```
Consistency_Check(dx)
   Begin
    If a node is server node    then
      Send a push message with update of Dx to nearest node in push-set

    If a node is caching node    then
     validx = false;
     If ($TTL_x$ > 0)          then
         $valid_x$ = true;              // the copy is valid
     else
          send validation request message to data server
          receive validation update message to update the cache copy and TTL
          $valid_x$ = true
  When a push message receives by the current node
    If node ID in the push-set     then
          Remove its ID
          Reply acknowledgment push message
          If push-set not empty     then
                Resend the push message to the nearest node in the set
          Else send acknowledgment push message to the data server
  Else when a node not in the push-set
          Update the cached copy of $D_x$ and $TTL_x$
          Forward the push message to the next hop on the routing path
End
```

Figure 9. Description of the cache consistency strategy.

The next algorithm in Figure 9 illustrates the pseudo-code of the cache consistency algorithm use.

5. Experimental Evaluation

The NS3 simulator was used to conduct this study, as it is a suitable tool for networking research and education, allowing for protocol design, comparison, and traffic evaluation. To implement the MANET routing protocol, the ad-hoc On-Demand Distance Vector (AODV) was selected from the options available in the NS3 simulator [3]. The simulation area was assumed to be a fixed size of 500m×2000m, with a density of 80 to 120 nodes and a transmission range of 250m. The area was divided into equal-sized square clusters of 3 by 12 (X-axis by Y-axis) grids, with each cluster identified by a column-wise numbering system from 0 to 35. The size of each cluster was set to be less than or equal to r/sqrt (2), where r is the transmission range, ensuring one-hop communication between cluster nodes. The wireless bandwidth was set at 2 Mbps, with other parameters being similar to those used in previous studies. The Table 1 below summarizes the significant simulation parameters, which were modified during the simulation to analyse their effects.

Table 1. Topology of area, mobility node and model traffic.

Parameter	Default value	Range
Simulation area	500m×2000m	
Number of nodes	100 nodes	[80,120] nodes
Number of clusters	36 (3×12) clusters	
Node cache size	800 kB	[200 ,1200] KB
Mobility pattern	random way point	
Node speed	0-2 m/s	[2 ,20] m/s
Bandwidth	2 Mpbs	
Database size	2000 items	
s_{min}	1 kB	
s_{max}	10 kB	
Pause time	250 s	[60-300] s
TTL	2000 s	[500 ,5000] s
Mean request generate time	5 s	[2 ,50] s
Transmission range	250m	
Simulation time	60000 seconds	[40000,80000] s

6. Results and Discussion

The paper evaluates the efficiency of cache management by measuring two performance parameters: Cache hit ratio and average hop count.

- **Cache hit ratio**: is calculated to measure the efficiency of cache management. The cache hit is classified into four categories: local cache hit, cluster cache hit, remote cache hit, and location hit. The local cache hit occurs when the requested data item is found in the local cache. The cluster cache hit occurs when the requested data item is found in the cache of one cluster member when the request is sent to the D-clusterhead. The remote cache hit occurs when the requested data item is found in one intermediate node, and the location hit is also considered a remote cache hit since the data is collected from remote nodes. The cache hit ratio is defined as the ratio of the number of data items retrieved from the cache to the total number of requested data items. Here cache hit ratio (*hit*) includes local cache hit (hit_{local}), cluster cache hit ($hit_{cluster}$) and remote cache hit (hit_{remote}). If n_{local}, $n_{cluster}$ and n_{remote} denote the number of local hits, cluster hits and remote hits respectively, and req_{suc} denotes the number of successfully received data items, then hit_{local}, $hit_{cluster}$, hit_{remote} and *Hit* are expressed as:

$$hit_{local} = \frac{n_{local}}{n_{local}+n_{cluster}+n_{remote}} \times 100\%$$

$$hit_{cluster} = \frac{n_{cluster}}{n_{local}+n_{cluster}+n_{remote}} \times 100\% \tag{6}$$

$$hit_{remote} = \frac{n_{remote}}{n_{local}+n_{cluster}+n_{remote}} \times 100\%$$

$$Hit = \frac{n_{local}+n_{cluster}+n_{remote}}{req_{suc}} \times 100\%$$

- **Average hop counts**: used to reflect the time delay for the cooperative caching system, instead of calculating the time latency directly. For the same case parameters, different routing protocols can result in different time latencies, thus making the time delay unstable and routing protocol problems are beyond the scope of this research, which focuses on the number of covered Hops through data requests that generally depend on where the requested item is found. The hops count between the caching node and the requester is expressed as:

$$Hop_{avg} = \frac{\sum hop\_length\_success\_req}{req_{suc}} \quad (7)$$

As shown in Figure 10, there is no obvious relationship between item size and cache hit ratio for Cache-Location strategy. If the item size is a small one, Cache-Data performs better than the proposed strategy, because Cache-Data has a higher cluster and remote cache hit ratio since it caches data for other nodes. Particularly when the size of the data item is small, more data can be cached in Cache-Data strategy and its cache hit ratio is significantly higher than that of the proposed strategy which depends on item size and TTL values to cache the data item. When the item size is large, the hybrid proposed strategy could cache more data items and locations for other nodes. This leads to achieving a high cache hit ratio. As a result of the high overall cache hit ratio, in comparison with Cache-Data, the proposed strategy achieves best results with an average improvement of 9.7%.

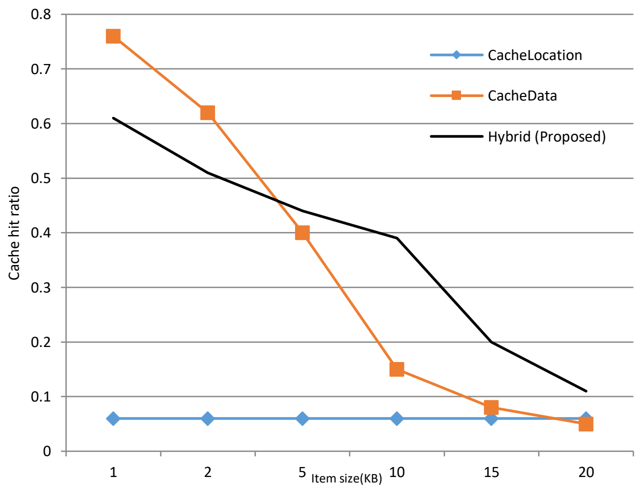


Figure 10. Plot of cache hit ratio versus different item size.

This shows, in particular, the strength of the strategy as it offers the best performance simultaneously. The proposed strategy works better than other strategies because of the high cache-hit ratio, as shown in Figure 11 when the cache item is small; Cache-Data has higher average hop count because its local cache and cache of cluster members contribute to reducing hop average. When the cache item is large, the proposed strategy performs better than others because it has higher cache hit ratio due to the hop count of local data hit is 0 and the average hop count of a remote cache hit is lower than that of other strategies. Comparing these three strategies, you can see that proposed strategy performs much better than Cache-Data or Cache-Location because both different schemes apply; data cache and location cache to different data items, taking advantage of both, with an average improvement of 2.5% as shown in Figure 11.

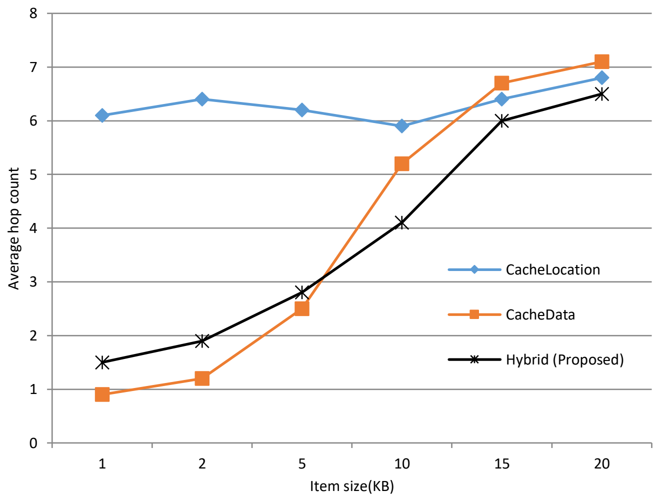


Figure 11. Plot of average hop count versus different item size.

From Figure 12, you can see that the proposed replacement policy based on calculated value for each cached item performs much better than Least Recently Used policy (LRU). As the cache size increases more data can be found in local and cluster caches, the need to access the remote and global cache has therefore been relieved. The hop number of the cluster data hit is one and lower than the average hop number of the remote data hit, so the cache hit increases. Since the cache size is large enough, most data items from the local and cluster cache can be accessed by mobile nodes. So it increases the cache hit and reduces the request latency. Comparing these two policies, the proposed approach when based on calculated value performs much better than when based on LRU. Due to the high hit ratio, value-based makes the best performance in comparison to LRU at all cache sizes and with an average improvement of 22%. The simulation varies between 80 and 120 mobile nodes in the area of the network to study performance in various node densities. As demonstrated in Figure 13, the value-based replacement has a better local hit ratio than the LRU policy at all the node densities. The number of mobile nodes in a cooperation cluster increases when the node density is high, this leads to an improvement in the cache cluster hit ratio and remote hit ratio. Value-based also performs better than LRU in terms of cluster and remote hit under different node densities. This can be explained by the fact that value-based considers various factors to make a more intelligent replacement decision.

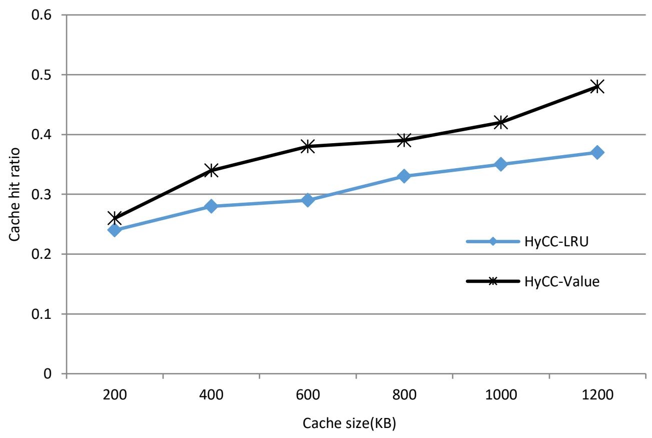


Figure 12. Plot of a cache hit ratio versus different cache size.

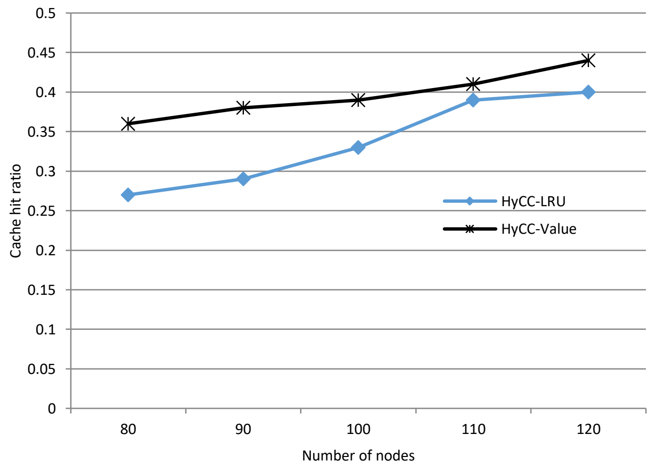


Figure 13. Plot of a cache hit ratio versus a different number of nodes.

The data server node by push modifies the cache nodes for the data update, which only impose unidirectional maintenance costs as overhead traffic and latency requests. The proposed approach, therefore, presented an effective push mechanism to reduce consistency maintenance costs. The data server node selects the caching nodes, which should be updated in one set. This experiment compares the Stateless Synchronous based approach (SS) with the proposed strategy. In the SS approach, the server periodically broadcasts invalidation reports to all the nodes in its access range. From the evaluation results, as shown in Figure 14, it finds that, even in comparison with the proposed consistency strategy, the SS strategy is more cost-effective, particularly when there are a large number of cache nodes. As the number of nodes increases, the caching node increases and the strategy's performance degrades. The selective push mechanism which presented in the proposed model that better performance is achieved based on cache status accounts in terms of average hop count which leads to better message overhead.

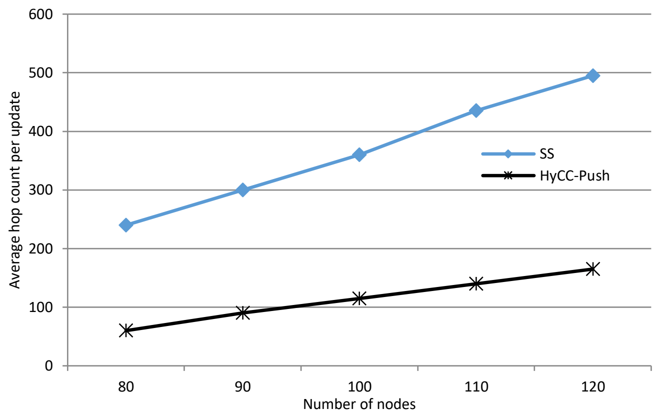


Figure 14. The average number of hops per update versus the different number of nodes.

7. Conclusions

In conclusion, the paper presents a hybrid cooperative caching strategy in MANETs that acts as a middleware layer between application layer and routing layer protocols to efficiently share cached data items among various ad-hoc mobile nodes. The proposed cache admission control algorithm assists in deciding which data items or location of data item can be cached for future use and helps in discovering and returning the requested data items from the neighbouring nodes of the requesting node. The evaluation of the proposed model shows promising results in terms of cache hit ratio and average hop count, which indicates the effectiveness of the proposed approach.

Overall, this research addresses a critical problem of cache management strategies in MANETs, and it is expected to contribute to the development of efficient caching strategies to improve the performance of MANETs. Future research can focus on the development of new caching strategies and the assessment of other protocols with important functions in MANETs. This will help in further improving the performance of MANETs and making them suitable for various real-world applications.

As a future work, it is recommended to develop and evaluate new caching strategies that take into account additional factors such as energy consumption, node mobility, and network topology to improve cache performance in MANETs.

References

[1] Abdullah A., Ozen E., and Bayramoglu H., "Enhanced-AODV Routing Protocol to Improve Route Stability of MANETs," *The International Arab Journal of Information Technology*, vol. 19, no. 5, pp. 736-746, 2022. https://doi.org/10.34028/iajit/19/5/5

[2] Cao J., Zhang Y., Cao G., and Xie L., "Data Consistency for Cooperative Caching in Mobile

Environments," *Computer*, vol. 40, no. 4 pp. 60-66, 2007. DOI: 10.1109/MC.2007.123

[3] Chandan R., Kushwaha B., and Mishra P., "Performance Evaluation of AODV, DSDV, OLSR Routing Protocols Using NS-3 Simulator," *International Journal of Computer Network and Information Security*, vol. 10, no. 7, pp. 59-65, 2018. DOI:10.5815/ijcnis.2018.07.07

[4] Chang N. and Liu M., "Revisiting the TTL-based Controlled Flooding Search: Optimality and Randomization," *in Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, Philadelphia, pp. 85-99, 2004. https://doi.org/10.1145/1023720.1023730

[5] Chauhan N., Awasthi L., and Chand N., "Global Cooperative Caching for Wireless Sensor Networks," *in Proceedings of the World Congress on Information and Communication Technologies*, Mumbai, pp. 235-239, 2011. DOI: 10.1109/WICT.2011.6141250

[6] Du Y. and Gupta S., "COOP-A Cooperative Caching Service in MANETs," *in Proceedings of the Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services-(icas-isns' 05)*, Papeete, 2005. DOI: 10.1109/ICAS-ICNS.2005.37

[7] Dziyauddin R., Niyato D., Luong N., Atan A., Izhar M., Azmi M., and Daud S., "Computation Offloading and Content Caching and Delivery in Vehicular Edge Network: A Survey," *Computer Networks*, vol. 197, pp. 108228, 2021. https://doi.org/10.1016/j.comnet.2021.108228

[8] Gunasekaran R., Divya V., and Uthariaraj V., "Mitigating Channel Usage in Cooperative Caching for Mobile Ad Hoc Networks," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 7, no. 2, pp. 87-99, 2011. https://doi.org/10.1504/IJAHUC.2011.038995

[9] Gupta S. and Sharma T., "Cooperative Data Caching in MANETs and WSNs: A Survey," *in Proceedings of the International Conference on Intelligent Computing, Instrumentation and Control Technologies*, Kerala, 2017. 10.1109/ICICICT1.2017.8342787

[10] Jain D., Sharma S., Yadav S., and Singh J., "A Detailed Survey and Comparative Study of Cooperative Caching Methods for Mobile Ad Hoc Networks," *International Journal of Sensors Wireless Communications and Control*, vol. 9, no. 3, pp. 314-329, 2019. 10.2174/2210327908666181120103838

[11] Joy P. and Jacob K., "A Key Based Cache Replacement Policy for Cooperative Caching in Mobile Ad Hoc Networks," *in Proceedings of the 3rd IEEE International Advance Computing Conference*, Ghaziabad, 2013. DOI: 10.1109/IAdCC.2013.6514255

[12] Mohseni S., Hassan R., Patel A., and Razali R., "Comparative Review Study of Reactive and Proactive Routing Protocols in MANETs," *in Proceedings of the 4th IEEE International Conference on Digital Ecosystems and Technologies*, Dubai, 2010. DOI: 10.1109/DEST.2010.5610631

[13] Ramphull D., Mungur A., Armoogum S., and Pudaruth S., "A Review of Mobile Ad Hoc NETwork (MANET) Protocols and their Applications," *in Proceedings of the 5th International Conference on Intelligent Computing and Control Systems*, Madurai, 2021. DOI: 10.1109/ICICCS51141.2021.9432258

[14] Salem A., Alhmiedat T., and Samara G., "Cache Discovery Policies of MANET," *World of Computer Science and Information Technology Journal*, vol. 3, no. 8, pp. 135-143, 2013.

[15] Zhou X., Zou Z., Song R., Wang Y., and Yu Z., "Cooperative Caching Strategies for Mobile Peer-To-Peer Networks: A Survey," *Information Science and Applications*, pp. 279-287, 2016.

Amer Abu Salem holds a B.E, M.E and Ph.D. in Computer Science. He has 22+ years of teaching experience and is currently working in a dual role as a Director of Computer Center/ Associate Professor of Department of Computer Science, Zarqa University. His research interest includes Wireless and Mobile Computing, Cloud Computing, Information Security, Data Science and Artificial Intelligence, He has published 20+ papers in peer reviewed journals and conferences. He is a life member of many professional societies.