# A NEW MISSION ORIENTED ROUTING IN DELAY-TOLERANT NETWORKS BASED OPPORTUNISTIC DISTRIBUTED CACHING

**KUNKA BHEEMALINGAPPA**
Research Scholar,
Sri Krishnadevaraya University,
Ananthapuramu, Andhra Pradesh
kunkabheemaforever@gmail.com

**Dr. M. NAGENDRA**
Professor & Head, Research Supervisor,
Department of CSE,
Sri Krishnadevaraya University,
Ananthapuramu, Andhra Pradesh
m_nagendra@rediffmail.com

## ABSTRACT:
*In this paper, a new caching scheme has been proposed which takes into consideration mission-oriented applications of Delay-tolerant Networks (DTNs) such as in Military. In such applications data need to be cached following different criteria and priority levels. In these applications, other than data popularity based on request frequency, the importance is also given to who created and ranked POIs (point of interest) in the data (images), when and where it was created; higher rank data belonging to some specific location may be more important though the frequency of those may not be higher than frequency based popular data. Thus, our caching scheme for DTNs is designed by taking different requirements into consideration for defense applications so that access latency for higher priority but lesser accessed data is reduced. The performance evaluation shows that our caching scheme based on criteria other than popularity reduces the overall access latency, cache miss and cache memory when compared to other caching schemes.*
*Keywords: Delay-tolerant Networks, caching scheme, mission-oriented military network.*

## I. INTRODUCTION
Caching data within Delay Tolerant Networks (DTNs) presents many challenges [1,2] in mission-oriented military applications. DTNs are, by design, volatile as certain nodes may not be available (disconnected or not reachable) all the time in the network. In mission-oriented military applications, how individual nodes, and indeed the network at large should choose what data to cache, where to cache, and how much to cache and thereafter, on what basis to do cache replacement, are most important issues. Cached data within DTNs is discovered as means for (1) accessing data locally available in close proximity to reduce latency and accuracy/consistency, and (2) to reduce resource usage (bandwidth and energy) through a reduction in hops to the high priority data sources. DTNs present an even more significant challenge for data availability as they are, by nature generally isolated and disconnected from a larger network. Therefore, DTNs must cache mission-oriented critical useful data to share at the strategic locations in DTNs. The simplest solution to cache everything at every DTN device and update it as we make connections to other nodes in the network will increase collisions as nodes will duplicate data, and waste buffer space and energy. Situations can arise where specific nodes within the network become over utilized by caching too many data points due to the uncertainty of data transmission, thus, multiple data copies need to be cached at different locations to ensure better data dissemination. The difficulty in coordinating multiple caching nodes due to the lack of persistence network connectivity in DTN makes it harder to optimize the trade-off between data accessibility (cache proximate to the nodes) and caching overhead (energy utilization to disseminate the data across many peers). Thus, each node must decide what data to be cached as nodes cannot design a cooperative caching schemes [1,2,7] because of not knowing which other nodes they would meet.

### 1.1. Motivation
From DTN caching schemes we have studied, file popularity [2], social-relationships [4] and cooperative caching [1,7] have always been the main motivation behind the file selection for caching to satisfy future requests and how cache replacement should be done.

Although these methods are efficient in many general scenarios where users transfer picture or other kinds of media files in networks using mobile devices, there are some applications where these methods entirely fail to address the caching requirement.

Consider mission-oriented networks or Military applications where data that need to be cached must follow different criteria and priority levels. For example, some data despite of low access frequencies (only few nodes may access) should be cached with a highest priority (as those nodes which access data are decision-makers) at strategic locations which are completely different from the conventional DTN caching methods. The data caching of some classified data even with the increasing demand, should be restricted to a certain number (may be within a certain radius) to prevent possible data theft or to deal with other security issues. In addition, some data may not be as popular (based on only access frequency), but latency to access when needed may be such a concern that it must be accessible quickly. Thus, data popularity can be defined not only on the basis of request frequency, but based on who created and ranked those data, when and where they were created; higher rank data belonging at some specific location may be more important though frequency of those may not be high at this time than lower rank data. The following elements: location, access latency, data creator's ranking, and popularity of the data based on ranking by intermediate nodes will be the primary qualities use to judge data in how exactly to cache it. Caching can consider the location/rank of the node who created it, resources at that location, perimeter of the nodes, POIs (point of interest) in the data (images), and requests by different nodes and then predicting the caching needs at the next time instant and possible locations. Thus, a new caching technique for DTNs needed to be designed which takes these different requirements into consideration to fulfil the necessity of military applications in DTN networks

## II. RELATED WORK

Below, we review some of the caching schemes which are closer to our objective. What data to cache? The problem of selecting the data to be cached is addressed in [2, 4]. In [2], authors proposed a Selective Pushing algorithm which caches data using the Zipf-like distribution. The authors assume that the network is a homogeneous environment where all users share the same mobility or centrality statistic. In [4], they compute the content popularity (relative to the current node) by considering both the frequency and freshness of content requests arriving at a node over a history of request arrivals. Then they select the most popular data to cache at a network location. But whenever there is a change in cached data all the cached nodes have to update data, the social tie and digest tables which consumes time and energy. This article focuses on forming interest groups, which are sets of nodes interested in a data item, which is closer to real time scenarios. We consider this approach in our design for selecting data to be cached with less overhead during updates. For example, in DTN, updates at the friendly nodes at few hopes away from the creator are faster and easier, so nodes which experience more frequent updates should be closer to the owner who created.

Where to cache it? This problem is addressed in [1, 4, 5]. In [1], nodes are selected to be Network Central Locations (NCLs) based on three approaches. These NCLs will be used to cache popular data that is needed by the nodes in the network. When NCLs buffers are filled, it will start caching the data on the nearby reliable nodes. Coordination between different NCLs and optimization of data of every NCL can become an endless optimization cycle which consumes a lot of energy if the network is very big and can never be optimized. In [4], clusters/groups of nodes are formed as interest groups. The POIs of

the nodes in those groups are cached at the NCLs nearby to those groups of nodes. In [5], node groups are formed based on the frequency of contact between the nodes, and the nodes are merged or resigned from the group if the POIs of the nodes in a group changes

How much to cache? [3] does a good job of showing the performance differences between optimal and opportunistic 'Edge' caching. The results show that opportunistic caching performs on par with the optimistic caching in all most all the scenarios. We will be using opportunistic caching in our model to cache data at the Edge of the DTN network i.e., near the end user. This will save the memory across the nodes in the network avoiding unnecessary data caching

## III. MISSION-OREINTED NETWORK MODEL

The mission-oriented military network systems consist of tactical vehicles such as Warfighter Information Network-Tactical (WIN-T) or High Mobility Multipurpose Wheeled Vehicle (HMMWV) which act as Network Central Locations (NCLs). These NCLs are always active and they have long-range wireless connections to each other and individually, they may also have secure and reliable satellite connection. We assume here that DTN behavior is observed by the ground troops, and Humvees, and control base can also have more like MANET connectivity using Wi-Fi.

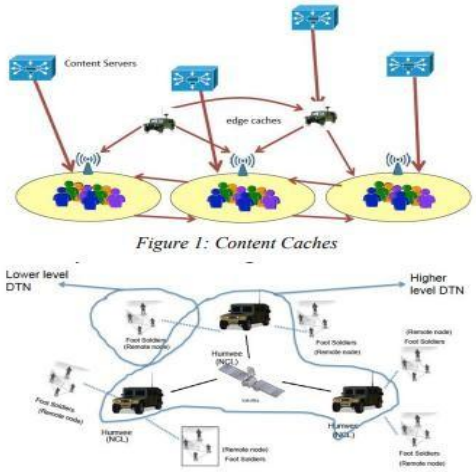


*Figure 1: Content Caches*

**Figure 1:** Network Model

**Content Caches:** The network consists of DTN devices to cache data and move in the network independently. These devices can range from tactical vehicles to custom-built systems just for caching. Some of these devices are mobile and some may be stationary. These devices can communicate with each other and exchange cached content between themselves using different wireless communication medium like Bluetooth, Wi-Fi direct, etc.

These devices will transfer data between them based on the demand/request around the nodes nearby. Some of these devices act as NCLs in the network and the nodes nearby can request cached data from others, and then cache data of their own. This military network system consists of mobile DTN nodes, which could be combat vehicles or ground forces, which move across the entire network freely. These ground forces need to send/receive messages which each other or should be able to send messages to tactical bases frequently that they are active and should be able to communicate with other tactical bases for further.

## IV. CACHING USING CONTENT CLASSIFICATION

In our case data needed to be cached belong to a mission-oriented network. We need to deal with two main problems: (1) how to define the method for caching by considering different requirements of military applications, and (2) a method to follow to replace the cached data when the cache memory becomes full. For caching, depending on the situation and requirement of the application, the data can be classified into different categories with different kinds of priority and ranking.

There exists a lot of data that might not be popular based on frequency of access but very important for the specified users like for the decision-makers (high rank nodes). These kinds of data should be cached at nodes, even if the popularity is quite low in the network currently, at some specific

locations from where data can be retrieved faster and data is secure/available (not far from the nodes making strategic decisions). Some data items may have very high priority as defined by the ranked nodes and its overall impact is adjusted by the combination of rank and priority. We define a time and spatial decay functions to classify data items and use the same to replace them when cache memory is full. Each node in the network will have a set of data items {(Wi, l(vi) | i א I}, where for each item i א I , Wi > 0 (varies between 0 and 1) is the weight of the item and l(vi) א V is the location of the item cached at node v, where V is a set of nodes in the network. Here, I is the set of data items in the network. The node defines weight of a data item where the item resides based on some parameters (including the space needed and considering all other data items it hosts) using the formula given below.

We denote by DIST(l(v), l(u)) the distance (number of hops) between the location of the two nodes v and u in the network. A decay function is a non-increasing function and it determines the weight of a remote item as a function of its distance and some varying attributes, such as time or number of requests to access the items. The spatial weight of an item i א I at the node v as viewed by a node u א V is

$$d_{uv} = DIST\big(l(v), l(u)\big)$$

The weight of an item of interest at location $v$, $w_v^i$, might be modelled as attribute values, $w_U^i$ associated with other location $u$, weighted by the inverse of the distance separating locations $u$ and $v$, $d_{uv}$ raised to a power $\beta$ and the difference between the current time T and the time of creation of the item $t_i$ and $t_i \leq T$ :

$$w_v^i = r_u^i * p_u^i * \frac{w_u^i}{(d_{uv}^i)^\beta (T - t_i)^\alpha}, \beta \geq 0, \alpha \geq 0$$

The exponent ù has the effect of reducing the influence of other locations as the distance to these increases. With ù=0 distance has no effect, whilst with ù = 1 the impact is linear. Values of ù >>1 rapidly diminishes the contribution to the expression from locations that are more remote. The exponent Į has the same effect as ù. There will be some threshold defined for to make a decision for cache replacement of an item, and the space is available. Note that the weight takes into consideration a priority p (a value of high (0.8), medium (0.6) and low (0.4) between 0 and 1) based on the category of data items defined by the ranker r (a value of high (0.9), medium (0.6) and low (0.3) between 0 and 1). Rank defines the importance of the node (0.9 means most important) and priority defines the importance assigned to the data item (0.8 being most important) by that node. The rank of a node r is fixed, where as priorites p change based on the node who has assigned the p value. r values do not decay but p decays with distance and/or time as they are defined by the creator. Therefore, the parameters ù and Į can be adjuted to reflect data items defined by high ranked nodes, and thus, will be decaying slowly with respect to the freuqency of access and time. Therefore, as explained earlier, due to slow decay, some items created by high ranked nodes may be cached as needed even if they do not have very high access frequency or created much earlier

We can use the same decay equations to calculate the decay in weight based on other attributes like number of requests for an item instead of time. The function will be as follows

$$w_v^i = r_u^i * p_u^i * \frac{w_u^i}{(d_{uv}^i)^\beta (Nt_1 - Nt_2)^\alpha}, \beta \geq 0, \alpha \geq 0$$

Nt2 and Nt1 are the number of requests received at timestamps t2 and t1. The exponent Į behaves the same way as it did for time dimension. It will make this function relevant for some of the classes of data items in the network.

We calculate spatial-time/request decaying weight sum for an item, i, using the following for a group of nodes g, which is a subset of all the nodes, V, in the network.

$$w_g^i = \sum_{v \in g} r_v^i * p_v^i * w_v^i, \forall g \subseteq V$$

We categorize the data in the network into five different categories.

**Class 0**: Emergency Data (high rank, high priority and high access frequency)

The emergency data broadcasts are quite rare but are the most important data. This kind of data can be classified as urgent with much higher priority. This data should be stored at each node that is connected to the network to provide instant access to that data, but its expiry will be much faster. In case of low buffer space, the data that needs to be removed is chosen by priority. First the least important data is removed and if space is still needed, then somewhat important but less frequently used data is removed followed by important and more frequent data. After the end of emergency broadcasts, all the data deleted should be restored from the neighboring nodes. An example of emergency data is a sandstorm report for next 2 hours or hazardous conditions marked in area maps.

The Class 0 data items decays exponentially with time and there may be no spatial decay for these data items. Therefore, the exponent ù will be 0 for these items and the exponent Į will be 2. The priorities will be 1 for this class of data items, and rank depends on the importance of node who has created the item and given its rank value between 0 and 1. As a result, if a node with high rank has created the Class 0 data item then they will be cached everywhere in the network until the value of the item decays with time/frequency reaches some pre-defined threshold. After that items will no longer be cached in the network or declassified to the lower classes as explained below.

**Class I:** Highly Important (both high to medium ranked and priority) and frequently used data

There are types of data like device log files, artillery count, injured-list, location-images for situational-awareness etc. that are important for certain decision-making nodes (e.g., military commanders/doctors) and are frequently accessed by only some selective nodes and updated, and to make sure that nothing gets stolen or lost from the network. We assume that soldiers and army vehicles act as nodes, where these kinds of files are accessed by everyone to inform other troops of their status, safe-routes, or how many soldiers got injured and their current status. Thus, these kinds of data need to be replicated at most of the nodes (in case if some group of nodes are not well connected). The need and priority of this data comes from the importance (ranking and priority) but it is less frequently usage data.

The disadvantage with these data items is to maintain data consistency; all the file copies at all the nodes need to be frequently updated and therefore, the access latency should be minimum. Frequent updating at few hops away drains battery power of the intermediate nodes, which might become crucial for the ground troops to make emergency contacts when needed. Therefore, to minimize these drawbacks, the data should be stored within a few hops from the source called cached-radius, as well as update time limit called update-frequency, and this update-frequency is dynamic within the cached-radius to best fit the situation. These data can consider POIs, and their quality as defined by the image quality and associated tags. Data, if stored at far reaching nodes may get the updates slowly and therefore, may be of poor quality in terms of POIs. The Class I items decay both with space and time. As mentioned above, access latency needs to be minimum for these items. We should try to cache the items near the POIs which will be selected based on the spatial weight sum. Their weights will decay exponentially with space but only linearly with time. Thus, the exponents ù and Į will be 2 and 1, respectively

**Class II:** Important and secure (high to low rank and priority) but less frequently used data

Many data which are classified, with limited access restriction comes under this

category. These files are not accessed all the times since they have limited access thus, make them less popular and are mostly accessed from limited locations. However, caching these kinds of files are very important because they need to be accessed without much latency. Thus, these data should be stored at NCLs and the data copies can be sent to other nodes upon request. These highest priority files can consume more buffer space like area maps for combat operations, POIs represent aggregated data, and thus, trying to replicate these files at every node reduces the caching efficiency of the network as well as can put the mission in danger. Therefore, trying to store these data at many nodes may pose a security threat. Thus, NCLs should be chosen well which has better caching space and has the best connectivity with other nodes. This kind of data ranked higher and has the highest priority of access by decision-making nodes. This data may be used for example in taking important decisions such as path for troop movement, which qualifies as class 0 data.

The Class II items, just like Class I, decays both with time and space but linearly. Thus, the exponents ù and Ị will be 1 and 1, respectively. Weight sum will be used to choose the NCLs to cache, which includes both priority and ranking of the data items as well

**Class III:** Not very important (medium to low rank and priority) but frequently accessed data

This data that may not look very important from a broader perspective, but plays a key role during a battlefield situation. For example, information about some activities like local news, sudden sandstorm, forest fire, etc. that need to be shared immediately with many nodes. These data become insignificant with time. This problem is similar to caching using file popularity at a certain location. These kinds of data need to be replicated at certain locations where it is crucial even though the data originated at some other location. These data during caching is temporary and should be easily replaceable. The priority of this data is after the first two types of data, Class I and II

The Class III items just like Class 0 decays with time and don't use spatial weights. The time decay is linear and number of requests can be used to decide on caching the items or promoting the item to Class 0. Thus, the exponents ù and Ị will be 0 and 1, respectively. Here also ranking and priory can rebalance the values to take a decision to cache replacement.

**Class IV:** Not very high ranked data (low rank, and medium to low priority), but accessed on popularity (higher access frequency)

At last there is some data, which is not important for the military networks (low ranked) but needs to be stored for general purposes (medium to low priority). These kind of data is cached using normal caching methods like file popularity, least recently used data etc. The priority of this data is way below the other kinds of data to provide easy access to important kinds of data. The frequency of data updating is kept low to maintain the update costs with very minimum overhead to preserve the battery power.

The Class IV does not decay neither with space nor with time. The exponents ù and Ị will be 0 and 0, respectively. We can use the linear decay based on the number of requests to decide on caching the item on an NCL or a node in the network. The ranking of such data is not high though priority can be very medium to low as defined by nodes.

## V. CACHING ALGORITHMS

### A. Initial Distribution of Caching Data

When data is first created at the source, the data needs to be classified by the source as one of the five categories mentioned above. A source can be either a remote node or a moving NCL. If the source is a remote node, then the data is sent to all its neighboring nodes within its nearest NCL range for further validation. Additionally,

while the data is sent to the NCL via remote nodes, it can be re-classified by the NCL. The heuristic for how they re-classify is up to them; for example, if the data is an image, and some other important nodes are getting similar images with POIs, they may raise the classification level. The important aspect is that prior to validation, remote nodes can also classify based on their understanding. This can impact/help in NCLs decision, once data reaches there.

It is important to send this data to an NCL because they are expected to have a larger cache size and wireless communication range. Additionally, if there is a Humvee, they may have additional data to influence the classification of the data in question. For example, if a photo of ambulance was taken, and to a solider it may look like an ordinary ambulance. However, if that is stolen by the enemy, it suddenly has become far more important, and that was not possible without the availability of additional data that the base (and its operators) had.

After an initial validation by the NCL and all its remote nodes, if the average class data is classified as class I or class II, then the data is sent to other NCLs for further validation. If most NCLs classify the data as class I, then that data is cached at some of the well-connected NCL locations who have validated it. Restricting the replication of such kind of data prevents theft by enemy

If most of the NCLs categorize the data as class II, i.e., important and urgent, then this kind of data needs to be further evaluated using local popularity. For example, using the geographical location of this data's popularity. Finding local popularity and locally caching data is important since it is classified as urgent. To find the local popularity, the NCL nodes, which classified the data as class II, are selected and requested for local popularity evaluation. If there are N active local nodes for each NCL and if the data item gets more than N/2 positive responses then that data is popular in that place, and the data is cached in that NCL and some nearby local nodes depending on the hop distance between the nodes. Trying to avoid the involvement of remote nodes until the last step helps in reducing communication and preserving the battery. If the data is initially classified as class III or class IV by the source and/or its nearest NCL then the data caching follows normal caching methods such as file popularity and other popular methods. If the data is initially classified as class 0 i.e. the data as an emergency message, then the data is sent to all NCLs to cache and after caching the data is evaluated for classification. This process is like class I process but now we cache the data and evaluate it later.

```
Input: Data item created at remote node or a moving NCL
Output: Classify the input data item and cache it in various
locations based on its class.
Classify(data)
if source is not an NCL then
  Validate(data, source.NCL)
if data.classification is Class0 then
  Cache(NCLs, data)
if data.classification is ClassI then
  for ncl in source.NCLs
    Cache(ncl, data)
if data.classification is ClassII and #NCLs classifying data as
Class II > #NCLs/2 then
  Cache(source.NCL, data)
If data.clssification is ClassIII or ClassIV then
  Cache(source, data)
```

**Algorithm 1: Distribution of Caching Data**

**B. Cache Redistribution upon Requests**

Data is cached with the intent that it will be requested later, and if it is found at an intermediate node along the path to the request it can be returned faster, or immediately if locally cached. It is because of that the cache should be re-evaluated upon a fulfilled request.

```
Input: Data item requested by a node
Output: The input data item is either cached at the source of
request or cached based on popularity
if Cached(source, data) then
  return GetCached(source, data)
startTime = Now
request = BroadcastRequestFor(data)
// Time passes while waiting for request response from remote
node
endTime = Now
if endTime–startTime > data.maxLatency then
  Cache(source, request)
else
  CacheByPopularity(source, request)
```

**Algorithm 2: Cache Redistribution**

Once the data is cached after initial distribution, the data should be replicated based on further user requests. When a data item is attempted to be retrieved from a source, the source either has it cached and can return it immediately (latency = 0), or it must hop to other nodes to retrieve the item (latency > 0). The amount of time it takes to retrieve the items via hops is that data item's latency. If the latency priority is lower (meaning the priority is to retrieve that data item in less time that it took to first retrieve it), then we cache it locally. We send a request to multiple DTN nodes to get an item, and take the latency of the node that returned the data as the maximum latency for that item. We would update the latency on future requests to the item if it changes and cache it on different nodes if needed to provide the reduced access latency. In case the item requested already has the best latency then we will cache based on the popularity in the neighbouring nodes.

### C. Cache Replacement

If a new data item needs to be cached at a node and if the cache space becomes full then some cache data needs to be removed to make room for the new data item. Cache space is divided into 2 levels with dynamic space allocation at each level. One level of cache memory is used to store class I, class II and class 0 types data and second level of cache memory is used for storing class III and class IV data. There is no separate memory space for class 0 data since class 0 data is very rare when compared to other types of data and allocating one more level just for class 0 data might deteriorate the cache performance of the network. If cache space becomes full and the new data item to be cached is of class I or class II then data from level 2 is removed and is cached in the neighboring nodes. While removing level 2 data, data at level 1 is declassified and moved to level 2 making space for the new item in level 1. If the space is still not sufficient then level 1 data is removed and must be cached in the neighboring nodes. If the new data item is of class III or class IV then level 2 data is removed and cached in neighboring nodes, until we have enough space to cache the new item. If data of class 0 needs to be cached and if the cache space is full then level 2 data is removed to replace this data. Unfortunately, if level 1 is filled, then class 2 is removed and cached in neighboring node to accommodate class 0 data. Class 0 data decays faster with time and can be removed later to get back the class II data. Class I data is not removed because this data is critical and will always be cached at well-connected NCL nodes.

## VI. SIMULATION AND EXPERIMENTS

To evaluate our caching algorithm, we used the ONE, a Java DTN simulator [6]. We use the ONE's default epidemic routing algorithm and extend the ActiveRouter Class to cache messages as per our proposed algorithm. The ONE DTN simulator exposes many different options and variables to tweak in for different results, and most of the results are dependent upon the setup of the nodes. For all our simulations, we will use the same initial nodes with roads as paths taken from a subset of the real world New Dubai map.

```
Input: New data item

Output: New data item will replace the cached items either in
Level1 or Level2 based on its class.

if newData.classification ≥ ClassII then
  for data in node do
    if data.classification ≤ ClassIII then

    // some other node may want to cache it, but we need to
remove it
      BroadcastToCache(data)
      Remove(node, data)

    if cache.spaceAvailible ≥ newData.size then

      Cache(node, newData)
      break
  // end for
  if not newData in node then
    BroadcastToCache(newData)
```

**Algorithm 3:** Cache Replacement

The nodes are all chosen in a militaristic setting:

• 6 groups of 10-40 Soldiers each spread across the map.

• 4 Humvees starting on a random point on the map.
• A lone Base is in the center and Humvees will commonly drive through the area

Soldiers and Humvees move around randomly. All randomness throughout the simulation has the same initial seed so all runs will have the same "randomness". We have 6 groups of soldiers in the simulation. We increase the node count for each group in increments of 10. Soldiers in each group move around in a circle of 500 meters based on dense or sparse areas accordingly in different locations of the map. Soldiers are not tied to roads, but Humvees are. Bases do not move. All nodes simulate a wireless interface akin to Wi-Fi, with varying ranges based on node type. Soldiers generate the data. Humvees and Bases are intended to hold a lot of cached data as they do not have to be carried by mobile devices and can afford to have larger storage and computing power, so they act as NCLs. All simulations are run for 12 hours in simulation time. Messages are generated and requested randomly throughout the simulation. Messages have a random initial classification, and can be adjusted by remote nodes (Soldiers) and NCLs (Humvees and Bases) based on the decaying weights during the simulation. We let the simulation run for half time (6 hours), and then during the other half aggregate the latencies of fulfilled requests. We compared three versions of our own algorithm to two most commonly used caching schemes (cache by popularity and Least Recently Used) and a simulation run without any caching. The three versions of our algorithm are as follows.

**Classification**.

We classify the data items created on the network and is cached based on the class (as discussed in section 4) assigned by the source and remains same till the end of the simulation.

| Parameters | | Values |
|---|---|---|
| No of Soldier Groups | | 6 |
| Soldier Count per Group | | 10 - 40 |
| Transmission Speed | Soldiers | 5 Mbps |
| | Humvee | 10 Mbps |
| | Base | 15 Mbps |
| Transmission Range | Soldiers | 200 meters |
| | Humvee | 500 meters |
| | Base | 1000 meters |
| Cache Size | Soldiers | 1 GB |
| | Humvee | 10 GB |
| | Base | 50 GB |
| Simulated Time | | 12 Hours |

**Table 1:** Simulation Parameters

**Dual classication**

without weights. This version also starts with the classification like the previous but during the simulation uses cache redistribution and replacement based on our algorithm to move around items in the network.

**Dual classification with weights.**

This is same as the previous version of our algorithm but also includes decaying weights which are calculated as discussed in the previous section 4. These weights are used to declassify items during cache redistribution and replacement parts of our algorithm. For cache by popularity, we use the simple case that ONE simulator supports where more "popular" (P) messages have a higher P, where P = H * N, with H being the number of hops that message has travelled, and N being the number of times the node, deciding to cache, has seen that message. We also used ONE's default caching scheme such as Least Recently Used (LRU). By default, with LRU, the least recently used messages are evicted from the end of the cache until buffer space is created for the new messages. Finally, we also ran simulations using the same parameters without any caching to compare and see how a caching scheme affects the access latency in a network. We poll latencies, for all these 5 scenarios to see how the caching schemes compare in different mission critical situations. Note that our proposed dual caching scheme can be

integrated with schemes like in [2, 4,7] for environments other than military networks.

## VII. RESULT ANALYSIS

The results of the latencies have been presented into two tables, with Table 2 being the recorded average access latencies for 6 different caching schemes discussed above.

Table 3 has the percentage values of cache size used for 3 specific scenarios. All numbers measured are in seconds inside ONE DTN simulator and 5 runs of each combination is averaged

Table 2 seems very useful to draw conclusions about the improvement of access latencies between different caching schemes. We can see that the plot for access latencies of different schemes in Figure 4 at node count 10 per group. We can clearly see that the five caching schemes vs. no caching as a base line clearly have a significant speedup. This makes sense as with data cached at more places, more requests can be fulfilled quicker.

Once we examine the differences in the caching schemes, we can clearly see that our dual algorithm out performs all other caching schemes. There is a 58% speedup when we compare our algorithm with LRU and 12% speedup when we compare it with cache by popularity (which is used by most DTN algorithms). We can see a speed up of 20% from dual without weight and 23% from dual with weights when compared to simple classification. So, moving around the cached data based on popularity, and decaying weights shows a significant effect on the access latency. Only caching based on classification can sometimes be slower than caching with popularity, and that is probably due to our design, which prioritizes highly classified data. Another interesting observation is that although dual with weights version of our caching scheme expects retrieval of critical items much faster, but we only see around a 3% speedup over dual without weights (see Table 2). Next, we see how our algorithm will affect cache hit/miss and cache memory used by each node during the simulation. We can see the cache miss percentage plot in Figure 3 for different node counts for dual with and without weights. As the number of nodes increases, the decaying weight help the network to make room for the incoming new messages by declassifying the unwanted data items and caching the new messages on NCLs and Base. The algorithm without decaying weights will just keep only the highly classified data leaving no space for the newly arriving low class data. This will result in the increase of cache miss, there by increasing the percentage of misses. We can also see the average percentage of cache used on all the nodes in the network in Table 3 and plot in Figure 4. We can clearly see that dual algorithm with decaying weights uses less cache memory than the one without weights and popularity caching. Again, the declassification and removal of data items which decays with time and distance helps free up the cache space. Our dual with weights is using less than 50% of the cache space used by popularity caching scheme and 15% less than dual without weights. From the results above, we can clearly see our dual algorithm with or without weights performs much better than any other common caching scheme in an environment like military networks. By using decaying weights, we can achieve more consistent access latencies and improve cache hit/miss and reduce the total cache memory used.

| Soldier Count Per Group | Dual with Weight | Dual without Weight | Classification | Popularity | LRU | No Caching |
|---|---|---|---|---|---|---|
| 10 | 461.90 | 479.66 | 604.59 | 523.00 | 1119.59 | 2070.19 |
| 20 | 462.97 | 468.71 | 627.27 | 535.50 | 1155.85 | 2110.75 |
| 30 | 465.85 | 472.94 | 649.13 | 556.47 | 1209.96 | 2275.59 |
| 40 | 471.33 | 497.39 | 684.66 | 573.96 | 1314.75 | 2486.17 |

**Table 2:** Access Latencies across different algorithms for different node counts

| Soldier Count Per Group | Dual with Weight | Dual without Weight | Popularity |
|---|---|---|---|
| 10 | 41.77% | 54.37% | 81.34% |
| 20 | 40.82% | 49.79% | 76.92% |
| 30 | 35.30% | 46.60% | 72.57% |
| 40 | 31.10% | 45.71% | 64.49% |

**Table 3:** Cache size used percentage

## VIII. CONCLUSION

In this paper, we propose a new caching scheme to opportunistically caches data for mission-oriented Delay tolerant Networks. The basics of this scheme demands that highly classified data is also cached often across the network, even if it is not very popular, and lower classified items are cached based on popularity. Simulations using The ONE DTN Simulator show that our solution does lead to significant latency improvements for important data access, over other caching schemes in mission oriented networks. Finally, our caching algorithm performs best by mixing data classification, redistribution and replacement and decaying weights. Our scheme has lower cache miss ratio and consumes less cache memory compared to other caching schemes. In future work, we will implement this scheme using mobile devices.

## REFERENCES

*[1] W. Gao; G. Cao; A. Iyengar, M. Srivatsa. "Cooperative Caching for Efficient Data Access in Disruption Tolerant Networks" IEEE Transactions on Mobile Computing, vol. 13, no. 3, pp.611- 625, March 2014.*

*[2] Tiance Wang, Pan Hui, Sanjeev R. Kulkarni, Paul Cuff "Cooperative Caching based on File Popularity Ranking in Delay Tolerant Networks". CoRR, 2014.*

*[3] A. Dabirmoghaddam, M. Barijough, and J. Garcia-LunaAceves. "Understanding optimal caching and opportunistic caching at the edge of information-centric networks." in ACM Proceedings of the 1st international conference on Information-centric networking, pp. 47-56., 2014.*

*[4] Tuan Le; You Lu; Gerla, M., "Social caching and content retrieval in Disruption Tolerant Networks (DTNs)," International Conference on in Computing, Networking and Communications (ICNC), pp.905-910, 16-19 Feb. 2015.*

*[5] Cabaniss, R.; Madria, S., "Content distribution in DelayTolerant Networks using social context," in Wireless and Mobile Networking Conference (WMNC), 2014 7th IFIP, vol., no., pp.1-8, 20-22 May 2014.*

*[6] Ari Kernen, Jrg Ott, and Teemu Krkkinen. "The ONE simulator for DTN protocol evaluation". In Proceedings of the 2nd International Conference on Simulation Tools and Techniques (Simutools '09). ICST (Institute for Computer Sciences, SocialInformatics and Telecommunications Engineering), ICST, Brussels, Belgium, Article 55, 10 pages.*

*[7] Wang, R., Hajiaghajani, F., & Biswas,S. "Distributed caching in mobile networks with heterogeneous content demand". In 14th IEEE Annual Consumer Communications and Networking Conference, CCNC 2017 (pp. 172-178)*