Social Caching and Content Retrieval in Disruption Tolerant Networks (DTNs)

Tuan Le, You Lu, Mario Gerla
Department of Computer Science
University of California, Los Angeles
Los Angeles, USA
{tuanle, youlu, gerla}@cs.ucla.edu

Abstract—In this paper, we extend our previous work on content retrieval in Disruption Tolerant Networks (DTNs) to support cooperative caching. Our caching scheme can adapt to the unstable network topology in DTNs, and enable the sharing and coordination of cached data among multiple nodes to reduce data access latency. Our key idea is to cache data at cluster head nodes that have the highest social levels, and thus receive many content requests. Furthermore, to reduce the caching overhead at central nodes and to bring data closer to the content requesters, we use multiple caching nodes along the content request forwarding path. Lastly, we propose a new cache replacement policy that considers both the frequency and recency of data access. Simulations in the NS-3 environment show that our caching scheme significantly improves the performance of content retrieval in DTNs.

Keywords—Disruption Tolerant Networks; Cooperative Caching; Social Network Routing

I. INTRODUCTION

In Disruption Tolerant Networks (DTNs) [1], mobile nodes contact each other opportunistically. Due to unpredictable node mobility, it is difficult to maintain persistent end-to-end connection between nodes. Thus, the store-carry-and-forward methods are used for data transfers from source to destination. Node mobility is exploited to relay packets opportunistically upon contacting other nodes. A key challenge in DTN routing is to determine the appropriate relay selection strategy in order to minimize the number of packet replicas in the network, and to expedite the data delivery process.

Cooperative caching has been extensively studied in both wireline and wireless networks [2], [3], [4]. However, due to the lack of persistent network connectivity, conventional cooperative caching techniques are not applicable to DTNs. There are two challenges for caching in DTNs. First, due to unstable network topology, it is difficult to determine the appropriate caching location for reducing data access delay. Second, to enhance data accessibility and to reduce the caching overhead of a single node, multiple nodes can be involved for caching. Yet it is challenging to coordinate caching among multiple nodes.

Regarding content search and retrieval services, Information-Centric Network (ICN) has been drawing increased attention in both academia and industry. In ICN, users focus on the content data they are interested in. They do not need to know where content data is stored. Each content packet is identified by a unique name generally drawn from a hierarchical naming scheme. The content retrieval follows the query-reply mode. A content consumer spreads his Interest packets through the network. When matching content is found either at the content provider or intermediate content cache server, the content data will trace its way back to the content consumer using the reverse route of the incoming Interest.

Previously, we proposed a novel disruption-tolerant mobile ICN [5], which leverages social network routing to query and retrieve content in DTNs. However, we did not consider caching. We assumed that each request can be satisfied by the original content provider. The main motivation for avoiding caches in our previous work was to exercise tight control on the copies delivered and the recipients of such copies. This made revocation possible. In this paper, we relax the control on copies and recipients and allow intermediate nodes to cache copies. Namely, we extend our previous design by addressing caching related issues. We propose a cooperative, socially inspired caching scheme in which popular content data are cached at cluster head nodes. These are popular nodes that have the highest social level (i.e. highest centrality) in the network, and thus are storing and forwarding most content requests. Yet, due to limited caching buffers in the mobile nodes, we also consider distributing cached data along content query paths. Neighbors of downstream nodes may also be involved for caching when there are heavy data accesses at downstream nodes. That is, downstream nodes move some of their existing cached data to neighboring nodes to make room for new data. Finally, we also consider dynamic cache replacement policy based on both the frequency and freshness of data access.

The rest of this paper is organized as follows. Section II reviews the related work. Section III summarizes our previously proposed content retrieval method. Section IV describes the design of the caching scheme in detail. Section V presents the experimental results. Section VI concludes the paper.

II. RELATED WORK

ICN has attracted much attention from the research community. Recent studies have focused on high-level architectures of ICN, and provide sketches of the required components. Content-Centric Network (CCN) [6] and Named Data Network (NDN) [7] are two implemented proposals for the ICN concept in the Internet. Their components, which include Forwarding
between two nodes using the history of encounter events. How
briefly introduces our previously proposed content retrieval
relationship: social tie, centrality, and social level. This section
for content retrieval. We leverage three key concepts in social
paths.

distribute caching data along the content request forwarding
to address the caching overhead at high-centrality nodes, we
requests are destined. Second, we use a novel weighing func-
popular data at high social-level nodes to which most content
accessed by other nodes in the network.

et al. [16], Gao
identical. In [16], Gao
classes, such that users in the same class are statistically
computed policy in the absence of a central authority, and uses
[15] apply a distributed caching replacement based on users'
contact duration limitation on cooperative caching. Authors in
in the distribution of social-tie values. As in equation (2),
Jain’s Fairness Index mechanism [18] to evaluate the balance
social ties to all other nodes. For the distribution, we adopt
Jain’s Fairness Index is used to determine whether users or
applications are receiving a fair share of network resources.

Jain’s Fairness Index: balance = \( \frac{(\sum x_i)^2}{n \times \sum x_i^2} \)  \( (2) \)

The centrality metric is defined in (3), where N is the
encountered node count in the encounter table.

\[ C_i = \alpha \frac{\sum_{k=1}^{N} R_i(k)}{N} + (1 - \alpha) \left( \frac{\sum_{k=1}^{N} R_i(k)^2}{N \times \sum_{k=1}^{N}(R_i(k))^2} \right) \]  \( (3) \)

Here, \( \alpha \) (set in our experiment as 0.5) is a parameter decided
by the user according to the specific scenario and network
conditions.

C. Compute Social Level

Nodes that have similar centrality tend to have similar level
of contacts with other nodes and thus similar knowledge
on content providers. To reduce the forwarding cost of the
content query phase, we propose to group together nodes with
similar centrality into the same cluster. Interest packets are
only forwarded from one cluster to another cluster. There is
no Interest forwarding within a cluster. Each cluster represents
a social level in the network.

We employ Lloyd’s K-means clustering algorithm [19],
which is proven to have polynomial smoothed running time
[20]. In the K-means clustering algorithm, K is a parameter
determined according to the specific scenario and network
much each encounter event contributes to the social-tie value is
determined by a weighing function \( F(x) \), where \( x \) is the time
span from the encounter event to the current time. Assume
that the system time is represented by an integer and is based
on \( n \) encounter events of node \( i \). Then, the social-tie value
of node \( i \)'s relationship with node \( j \) at time \( t_{base} \), denoted by
\( R_i(j) \), is defined as in (1).

\[ R_i(j) = \sum_{k=1}^{n} F(t_{base} - t_{j_k}) \]  \( (1) \)

where \( F(x) \) is a weighing function and \( \{ t_{j_1}, t_{j_2}, ..., t_{j_n} \} \) are
the encounter time when node \( i \) met node \( j \) and \( t_{j_1} < t_{j_2} < \ldots < t_{j_n} \leq t_{base} \). We take \( F(x) = (\frac{1}{x})^{\lambda x} \) where \( \lambda = e^{-d} \) is a
control parameter, which has been shown in [17] to achieve a
good trade-off between recency and frequency.

B. Compute Centrality

Each node maintains a social-tie table that contains the
social distances from the current node to all other encountered
nodes. During the encounter period, the social-tie table is
exchanged and merged into the other node’s social-tie table.
Based on the social-tie table, a node can compute each other
node’s centrality. We estimate the centrality by considering
both the average social-tie values and their distribution.
Namely, we favor nodes with high, uniformly distributed
social ties to all other nodes. For the distribution, we adopt
Jain’s Fairness Index mechanism [18] to evaluate the balance
in the distribution of social-tie values. As in equation (2),
Jain’s Fairness Index is used to determine whether users or
applications are receiving a fair share of network resources.

III. BASIC CONTENT RETRIEVAL FRAMEWORK

In [5], we proposed a social based forwarding approach
for content retrieval. We leverage three key concepts in social
relationship: social tie, centrality, and social level. This section
briefly introduces our previously proposed content retrieval
scheme.

A. Compute Social-Tie Relationship

Two nodes are said to have a strong tie if they have
met frequently in the recent past. We compute the social tie
between two nodes using the history of encounter events. How

Information Base (FIB), Pending Interest Table (PIT), and
Content Store (CS), form the caching and forwarding sys-
tem for the content data. Several mobile ICN architectures
have also been proposed for the mobile environment, e.g.,
Vehicle-NDN [8] for the traffic information dissemination,
and MANET-CCN [9] for the tactical and emergency application.

Research on packet forwarding in DTNs originates from
Epidemic routing [10], which floods the entire network. Recent
studies develop relay selection techniques to approach the
performance of Epidemic routing with lower forwarding cost.
In [11], nodes are ranked using weighted social information.
Messages are forwarded to the most popular nodes (highly-
ranked nodes) given that popular nodes are more likely to
meet other nodes in the network. The explicit friendships are
used to build the social relationships based on their personal
communications. SimBetTS [12] uses egocentric centrality and
social similarity to forward messages toward the node with the
highest centrality, to increase the possibility of finding the
optimal carrier to the final destination. BubbleRap [13]
combines the observed hierarchy of centrality and observed
community structure with explicit labels to decide on the best
forwarding nodes. The centrality value for each node is pre-
computed using unlimited flooding.

Cooperative caching for mobile ad-hoc networks has also
been studied widely in recent years. Zhuo et al. [14] pro-
pose a social-based caching that considers the impact of the
contact duration limitation on cooperative caching. Authors in
[15] apply a distributed caching replacement based on users’
computed policy in the absence of a central authority, and uses
a voting mechanism for nodes to decide which content should
be stored. In their model, mobile users are divided into several
classes, such that users in the same class are statistically
identical. In [16], Gao et al. propose to intentionally cache
data at a set of network central locations which can be easily
accessed by other nodes in the network.

Our work differs from all these studies in several aspects.
First, we leverage the social hierarchy to efficiently cache
popular data at high social-level nodes to which most content
requests are destined. Second, we use a novel weighing func-
tion which takes into account both the frequency and freshness
of the content request to evaluate content popularity. Third,
to address the caching overhead at high-centrality nodes, we
distribute caching data along the content request forwarding
paths.

A. Compute Social-Tie Relationship

Two nodes are said to have a strong tie if they have
met frequently in the recent past. We compute the social tie
between two nodes using the history of encounter events. How
scalability. A larger K value will benefit the packet delivery ratio but cause higher transmission cost.

D. Content Name Digest Convergence

To facilitate content query, each content provider actively announces its content name digest (a list of names of contents a node owns) to nodes in higher centrality clusters. Each node maintains a local data structure called digest table (which maps the provider ID to the digest) to store the received digests from lower centrality nodes. Furthermore, when nodes encounter each other, the digest table will be sent to the node with the higher centrality. Throughout this process, the content name digests from each content provider are converged toward higher centrality nodes. Subsequently, higher centrality nodes have broad knowledge of which node owns which content in the network.

E. Interest Packet Forwarding

The Interest packet is carried by the requester and is forwarded to the first encountered node that has a higher social level than the requester itself. Subsequently, the requester keeps a copy of the Interest packet and forwards it to the next encountered node that has an even higher social level than the last relay node.

After a node receives an Interest packet from other nodes it encountered, it will first check its local digest table to see if there is any matched name. If no matched name is found, it will continue forwarding the Interest packet. Each relay node performs the same strategy; forwarding the Interest packet to the next relay node that has a higher social level than the last relay node. Following this strategy, the Interest packet is forwarded upward, level by level, toward the most popular node in the centrality hierarchy.

Since the content name digest is updated and converges toward higher social level nodes, if the content is present in the network, the Interest name will eventually match the content name in the digest table at high social level nodes. At this point, the content provider ID is disclosed, and the Interest packet will be social-tie routed (i.e., routed in a DTN mode) toward the content provider. In social-tie routing, the packet is forwarded to the newly encountered node only if the node has a higher social-tie value to the destination node in comparison to the current node.

F. Data Packet Forwarding

After the Interest packet reaches the content provider, the content provider will social-tie route the data packet back to the requester. The content provider only responds once to the same Interest packet that originates from the same requester. Subsequent received duplicate Interest packets are ignored.

IV. CACHING SCHEME

In this section, we first describe three prominent issues for any caching system: which data to cache, where to cache, and how to manage the cache (cache replacement policy). The ultimate goal is to maximize the cache hit rate. Subsequently, we present the caching protocol in detail.

A. Cached Data Selection

When the cache buffer space is free, a natural choice is to cache any data. When the cache space is full, it is more selective toward which data to cache. Intuitively, popular data is a good candidate for caching. We 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. Equation (4) defines the popularity of content i based on the past n requests to this content.

\[ P_i = \sum_{k=1}^{n} F(t_{base} - t_k) \quad (4) \]

In this equation, \( t_{base} \) is the current time, and \( t_k \) is the past arrival time of the request for content \( i \). We assume that the system time is represented by an integer and that \( t_1 < t_2 < ... < t_n \leq t_{base} \). We use a weighing function \( F(x) = (\frac{1}{2})^\lambda x^\lambda \), where \( \lambda \) is a control parameter and \( 0 \leq \lambda \leq 1 \). The control parameter \( \lambda \) allows a trade-off between recency and frequency. As \( \lambda \) approaches 0, frequency contributes to the content popularity more than recency. On the other hand, when \( \lambda \) approaches 1, recency has a greater influence on the content popularity than frequency. Following [17], to achieve a good trade-off between recency and frequency, we set \( \lambda = e^{-\frac{1}{4}} \).

B. Caching Location

If each node has unlimited cache space, then it is trivial to identify suitable caching locations, as data can be cached everywhere. Given that each node has limited space for caching, we follow a conservative approach and only cache data at nodes satisfying the following conditions:

1) Selected nodes are on the query forwarding paths.
2) They are traversed through by many common requests.

In social-based forwarding introduced in Section III, requests are forwarded upward, level by level, toward socially active nodes that have high centrality value, and thus have broad knowledge of content ownership in the network. Hence, the two conditions can be easily satisfied by cluster head nodes, which have the highest social level in the network. Furthermore, to ensure that cluster head nodes are not overloaded by too many requests and cached data, we further replicate and cache popular data at downstream nodes along the request forwarding paths. This will benefit requester nodes that are in close proximity to each other as the second will get the data requested by the first. Note that each node maintains its own local view of which data is popular based on how frequent and recent requests arrive at the node. Once the node determines that a certain data is popular, it will actively request the data for caching from the content provider (if it is a cluster head node) or from an encounter node (if an encounter node carries that data).

Caching in neighbors of central nodes, whose caches are heavily utilized, is another optimization implemented in this scheme. When a central node (typically, a cluster head node or any node along the popular request forwarding paths) cannot cache new data due to limited space, it will move some of
its existing cached data to neighboring nodes. Within the list of data that can be moved, the central node first moves more popular data to nodes with the strongest ties to the central node. We avoid moving data to nodes on the same forwarding paths, as the cache buffers of these nodes tend to be already heavily utilized. Query processing (i.e. cache lookup) is handled in the same order. That is, we first lookup the current node’s cache. If the data is not found in the cache, the current node propagates the query to higher social-level nodes and to nearby nodes to which it has the strongest ties.

C. Cache Replacement

When the cache buffer is full, existing data must be evicted from the cache, to accommodate new data. There are two related issues:

1) Determining the amount of data to evict.

2) Identifying particular data to evict.

For the first issue, we need to evict as much data as the size of the new data. Regarding the second issue, we propose to remove data from the cache that is identified as least popular. That is, we consider both the frequency and freshness of data access. As we will show in Section V, this replacement policy is superior to traditional cache replacement strategies such as Least Recently Used (LRU) or Least Frequently Used (LFU). In LRU-based caching, contents that were popular (but not often requested in recent times) tend to get evicted from the cache. This can lead to the eviction of popular contents when the temporal distribution of the requests to a content is not uniform. Similarly, LFU-based caching schemes do not perform well when the content pool is dynamic and the popularity of the contents in a cache decreases with time. By considering both the frequency and recency of accesses, we account for the temporal changes in content popularity.

D. Caching Protocol

Pseudocode 1 and 2 outline our caching protocol. In pseudocode 1, we assume that the Interest packet arrives at a node that does not already have the data (either cached data or owned data) that matches the request. In lines 15-16, the node actively asks the content provider for a data copy for caching at the current node only when the content popularity exceeds a threshold $\delta$. The threshold is used to avoid frequent data replication and forwarding overhead from the content provider to the current node.

In addition, to enable cooperative sharing of popular data, nodes exchange a list of cached and owned data in the form of content name digest upon encountering each other. If the cache candidates belong to the list, nodes request them from the corresponding encounter nodes. Nodes also periodically advertise their spare cache capacity to each other. This allows central nodes to opportunistically make decisions regarding which cached data to move to neighboring nodes so that central nodes have more space to cache new popular data.

Fig. 1 illustrates all the steps from when nodes request a content until the content is delivered and cached at intermediate nodes. We assume nodes request the same content.

Pseudocode 1 Handle Interest Packet Arrival

1: when an Interest packet is received
2: if there is enough free space then
3: mark the content as a cache candidate
4: else
5: re-evaluate the popularity of requested content & cached data
6: find cached data that are less popular than requested content
7: if evicting them creates enough space for the content then
8: mark the content as a cache candidate
9: end if
10: end if
11: check my local content table for the content provider ID
12: if there is a match then
13: social-tie route Interest packet to content provider
14: if the requested content is a cache candidate then
15: if popularity of requested content is higher than $\delta$ then
16: request content provider to replicate data to this node
17: end if
18: end if
19: end if

Pseudocode 2 Handle Data Packet Arrival

1: when a Data packet is received
2: if there is a cache candidate matching the data then
3: if there is not enough space in the cache buffer then
4: evict data that are less popular than cache candidate
5: end if
6: cache the received data
7: end if

1) Interest packets are generated and routed to the cluster head on two different paths using social-level routing.

2) Cluster head social-tie routes the Interest packets to the content provider. Assume the popularity of the requested content exceeds threshold $\delta$. Cluster head also requests the content provider to send it a copy of the content.

3) The content provider social-tie routes the Data packets to the two requesters and the cluster head. The content is then cached at the cluster head.

4) Additional Interest packets are generated and routed toward the cluster head using social-level routing.

5) Since the cluster head has a cache copy of the content, the cluster head social-tie routes the content to the requesters.

6) Nodes along the common request forwarding paths request a cache copy of the content from upstream nodes, and cache it locally.

7) An Interest packet is generated and routed toward higher social-level node.

8) Since the node has the cache copy, it can serve the request without propagating the request upward to the cluster head.

V. PERFORMANCE EVALUATION

In this section, we evaluate the performance of the proposed caching scheme in a packet-level simulation using a real world mobility trace. We first describe the simulation setup, followed by the metrics used and the results.
A. Simulation Setup

We implemented the proposed caching scheme using the NS-3.19 network simulator. DTN nodes advertise their Hello message every 100ms. In order to test the worst-case situation in terms of cache sharing and thus performance, we assume that each node contains unique content which is different from all other nodes. We also assume that the content data can be retrieved as 1MB in size so that the measurement will not be affected by the content size variance. We fix the content popularity threshold δ to 2.5. That is, we consider content as popular when it arrives at a node 3 times within 300ms interval. Finally, we assume that the caching buffer of nodes is uniformly distributed in range [10MB, 30MB].

We use the IEEE 802.11g wireless channel model and the PHY/MAC parameters as listed in TABLE I. To gain meaningful results, we use the San Francisco cab mobility trace that consists of 116 nodes and is collected in 1 hour in the downtown area of 5,700m x 6,600m. We fix the broadcast range of each moving object to 300m which is typical in a vehicular ad hoc network (VANET) setting [21].

We evaluate both the caching performance and the effectiveness of cache replacement. For the former, we compare our caching scheme CoopCache against NoCache scheme, in which caching is not used for data access and each request is only responded by the unique content provider. To observe the maximal benefit of caching, we let all nodes request exactly the same content. For the latter, we consider content replacement policy based on the content popularity against traditional LFU and LRU policy. In this experiment, each node requests random content in the network. For statistical convergence, we repeat each simulation three times.

B. Evaluation Metrics

We use the following metrics for evaluations:

- **Success ratio**, the ratio of content queries being satisfied with the requested data.
- **Average delay**, the average delay for getting responses to content queries.
- **Total cost**, the total number of message replicas in the network. This includes both Interest and Data packets.

C. Caching Performance

Fig. 2 shows the performance of NoCache scheme and our proposed CoopCache scheme. As we increase the simulation time from 100s to 1,100s, the success ratio of both schemes is improved, because data has more time to be discovered and delivered to requesters. However, the improved ratio is increased at a significantly faster rate in CoopCache scheme than in NoCache scheme. This is because CoopCache replicates and caches popular data at nodes close to the requesters, resulting in a higher hit rate and lower latency. For the same reason, CoopCache has much lower average delivery delay than NoCache. In NoCache, the majority of content queries need to be propagated to the cluster head node, and then social-tie routed to the content providers. The Interest forwarding step alone adds a significant delay to the overall content query and delivery delay in NoCache. CoopCache, by leveraging intermediate caching nodes along the common forwarding paths, can eliminate many of the delays from the NoCache scheme. Finally, in Fig. 2(c), NoCache suffers a very high cost. Content query and delivery in NoCache often traverses many hops, thus resulting in a large amount of Interest and Data packet replication. CoopCache, on the other hand, uses intermediate nodes for caching, thus shortening the Interest forwarding paths and lowering the overall cost of the system.

D. Performance of Cache Replacement

In this subsection, we evaluate the performance of our proposed cache replacement policy based on content popularity. We compare our policy against the traditional replacement policies including LFU and LRU. We fix the simulation duration to 600s. We vary the content size from 1 to 10MB and still assume that all contents are of the same size. This enables us to increasingly put more pressure on cache replacement to observe the effectiveness of different schemes.

The simulation results are shown in Figure 3. The popularity-based replacement scheme outperforms LFU and LRU policy on all three metrics. The performance gap grows bigger as we increase the content size. This is because when the content size is small, the cache buffer constraint is not tight, and therefore cache replacement is not frequently conducted. Subsequently, the performance difference is not too
significant. However, when the content size becomes larger, cache replacement is conducted more frequently, and LFU and LRU do not always select the most appropriate data to cache, due to improper consideration of content popularity. Thus, the advantage of our popularity-based scheme rises significantly when the content size is set to 10MB.

VI. CONCLUSION

We have proposed a new cooperative caching scheme based on the social relationship among nodes in DTNs. In this scheme, data is dynamically cached at selective locations in the network such as cluster head nodes, which have the highest social levels, and nodes along the common request forwarding paths. We have described a new cache replacement policy based on the content popularity, which is a function of both the frequency and recency of data access. Extensive simulation results show that our scheme significantly improves the ratio of successful queries, while reducing the delay and cost of message replicas.

REFERENCES