Array 19 (2023) 100299

Contents lists available at ScienceDirect

Array

journal homepage: www.elsevier.com/locate/array

A hybrid recommendation scheme for delay-tolerant networks: The case of digital marketplaces☆

Victor M. Romero II *, Bea D. Santiago, Jay Martin Z. Nuevo

Division of Natural Sciences and Mathematics, University of the Philippines Visayas Tacloban College, Magsaysay Boulevard, Tacloban City, 6500, Leyte, Philippines

ARTICLE INFO

MSC:
0000
1111

Keywords:
Delay-tolerant
Digital marketplace
Recommender systems

ABSTRACT

Recommender systems are widely-adopted by numerous popular e-commerce sites, such as Amazon and E-bay, to help users find products that they might like. Although much has been achieved in the area, most recommender systems are designed to work on top of centralized platforms that are traditionally supported by fixed infrastructure like the Internet. Hence, additional work is warranted to examine the applicability and performance of recommender systems in challenging environments that are characterized by dynamic network topology and variable transmission delays. This study deals with the design of a recommender system that is compatible in a delay-tolerant network where communication is supported by opportunistic encounters between participating nodes. The proposed approach combines collaborative filtering and content-based filtering techniques to generate rating predictions for users. To make the system more tolerant against interruptions, each node maintains a local recommender that generates predictions using user profiles that are obtained through opportunistic exchanges over a clustered topology. Simulation results indicate that the proposed approach is able to improve coverage while alleviating the cold-start problem.

1. Introduction

Delay-Tolerant Networks (DTN) are communication networks that are specifically created to operate in challenging environments lacking fixed infrastructure or where traditional networking is not feasible. These environments include isolated and remote areas as well as inter-planetary communication. In the absence of fixed infrastructures, DTNs leverage readily available wireless interfaces and opportunistic encounters between mobile participating nodes. Consequently, these networks exhibit characteristics such as high and fluctuating transmission delays, absence or failure of end-to-end paths, and significant node mobility [1,2]. In order to facilitate communication in such an unpredictable and highly dynamic environment, the network relies on a *store-carry-forward* mechanism to tolerate delay and achieve incremental progress until the message is received by the intended destination [3]. The process of how messages are propagated in a DTN is depicted in Fig. 1. When a source node needs to send a message, it first stores the message in its buffer until it encounters another node. If the second node meets the forwarding conditions and is capable of carrying the message, the source node forwards the message to the second node. The second node then carries the message until it encounters another node that can further forward it. This series of forwarding continues through intermediate nodes until the message reaches the destination node.

It has been proposed that electronic commerce can be supported effectively by delay-tolerant networking over an infrastructure characterized by high transmission delay [4]. This type of system could bring advantages to both consumers and entrepreneurs, even when compared to the traditional Internet. E-commerce, along with other Internet-based services like email, social network, and media streaming, naturally aligns with delay-tolerant networking because it does not require brief round trip time and is asynchronous. However, in order for e-commerce to be viable in a delay-tolerant environment, it is important to minimize the number of query/response interactions that occur while the user interacts with the server. To address this, more computational tasks must be transferred to the user devices, including relocation of code (such as rendering order forms) and data (such as shopping catalogs). By adopting this approach, users can effortlessly browse products and compare prices without being dependent on server availability. Moreover, they become immune to delays and interruptions caused by congestion in the network infrastructure. Opportunistic encounters in the DTN-based digital marketplace can be focused instead on the propagation of product information towards the incremental completion

☆ This study was made possible through the support of the University of the Philippines Visayas Office of the Vice Chancellor for Research and Extension (Regular In-house Research Grant).

* Corresponding author.
E-mail address: vmromero@up.edu.ph (V.M. Romero II).

https://doi.org/10.1016/j.array.2023.100299
Received 30 December 2022; Received in revised form 4 June 2023; Accepted 11 June 2023
Available online 15 June 2023

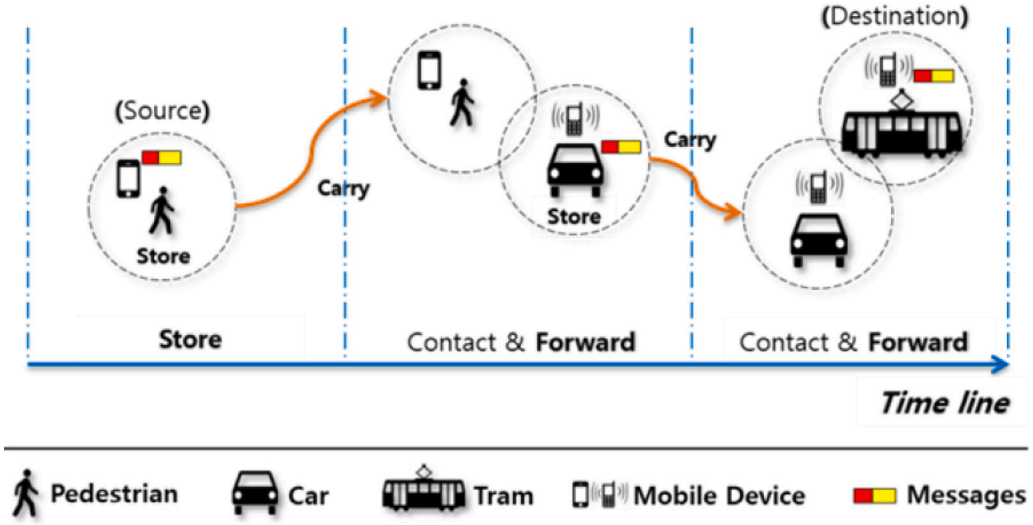


Fig. 1. Store-carry-forward mechanism in DTN taken from [3].

of locally stored product database and requests for order fulfillment. Although the envisioned system in [4] was not implemented, the author has highlighted key design considerations and advantages including exploitation of available computing power in end devices, reduction of server requirements, and potential to increase revenue by eliminating entry barriers. We independently developed a prototype of a DTN-based digital marketplace motivated by the potential of delay-tolerant networking in bridging the opportunity gap caused by the disparity in Internet access across different areas in the Philippines. The application was specifically designed for the Android operating system and utilized IBR-DTN's Application Programming Interface (API) [5] to convert user actions, such as product registration and chat, into bundles that could be distributed within the network. The application provided a consistent set of functions to all users, enabling them to simultaneously assume the roles of both buyers and sellers. By capitalizing on existing network interfaces and leveraging opportunistic encounters, the resulting delay-tolerant digital marketplace was able to operate without the need for additional hardware or network requirements.

Similar to traditional online counterparts, the number of available products in a DTN-based digital marketplace is expected to grow as more users join the platform. This growth is amplified by the fact that different sellers can list similar products in their individual catalogs. Eventually, the product database outgrows the capacity of human agents to browse through and compare products, which could hinder meaningful interactions such as commitment of transactions [6]. Several approaches have been proposed to address information overload in digital marketplaces. Each approach can be characterized using a common framework that is based on two variables: personalization and automation. One example of an impersonal and automatic approach is the utilization of best seller lists generated from transaction reports. On the other hand, there are digital services that offer personalized filtering options, allowing users to navigate through large databases based on specific criteria, but requiring manual input from the users.

1.1. Recommender systems

Recommender Systems provide a personalized and automated solution to address the information overload challenge. These software tools assist users in navigating overwhelming product catalogs by analyzing their preferences and recommending items that are more likely to be of interest to them. While recommender systems are commonly used in e-commerce, their application extends to various other fields such as health (e.g. predicting drug side effects [7]), social networks (e.g. recommending friends [8]), industry (e.g. recommending tourist destinations [9]), e-learning (e.g. recommending useful learning materials [10]), music (e.g. recommending music playlists [11]), Internet of Things (recommending IoT services [12]), and nutritional information system (recommending healthy recipes [6,13]).

1.1.1. Phases in a recommender system

A recommendation system functions by taking as input a collection of users with their interest profiles to produce a recommended subset of items for each user. This process has three stages, namely: (1) information collection phase, (2) learning phase, and (3) prediction/recommendation phase (see Fig. 2).

In the information collection phase, the system derives information that can be used to infer user interest [14]. This can be accomplished through explicit or implicit means. Explicit gathering involves asking users to provide specific details about themselves, including demographic attributes (such as name, age, and occupation) and personal preferences (such as hobbies). While this approach has the potential to yield reliable information for generating high-quality recommendations, the additional burden and privacy concerns may lead many users to skip or avoid this process entirely. Furthermore, due to its *snapshot* nature, this approach fails to capture the dynamic nature of user preferences over time. On the other hand, implicit gathering does not require users to perform additional tasks. Instead, interest profiles are automatically inferred by observing user behavior, such as purchase and navigation history, followed links, and time spent on specific items. Implicit gathering techniques are less burdensome for users, but they rely on a substantial amount of user–item relationships to accurately deduce user preferences.

In the second phase, known as the learning phase, the system processes the collected user information to extract relevant features or attributes. These extracted features are then utilized to construct user profiles using data mining techniques or machine learning.

In the final phase, known as the prediction and recommendation phase, the system employs the user profiles derived in the previous step to estimate the level of interest each user may have towards undiscovered items. Various filtering techniques can be employed based on the available information. One simple and straightforward method is to rank the items according to their predicted ratings and select the top N items with the highest predicted ratings from the list [14,15].

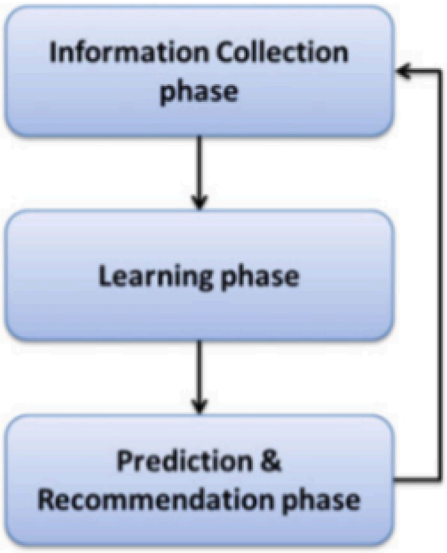


Fig. 2. Phases of recommendation process taken from [14].

1.1.2. Filtering techniques in recommender systems

Various techniques can be used in the prediction and recommendation phase of a recommender system. Among these approaches, two well-known methods are content-based filtering and collaborative filtering.

Content-based filtering recommends items by analyzing the properties of previously rated items to build individual user profiles [16]. The representation of a user's level of interest in a new item is determined by measuring the similarity between the user's profile and the features of the new item. Thus, the representation of user profiles plays a central role in content-based filtering.

The Vector-Space Model (VSM) is the simplest and most commonly used representation of user profiles in content-based filtering. The VSM is a statistical term-based technique widely employed in information retrieval literature. It represents the contents of documents as vectors of weighted terms. Similarly, user profiles are also represented as vectors of weighted terms that reflect the user's preferences. The weight assigned to each keyword signifies its importance in representing the document or the user profile.

The adoption of content-based filtering in a recommender system presents several advantages [17]:

- User independence: Content-based filtering relies solely on the characteristics of previously rated products to construct user profiles. Therefore, it does not require data from other users to make recommendations for a specific user.
- Transparency: Recommendations made by content-based filtering can be easily explained by examining the features that contribute to the inclusion of an item.

However, content-based filtering also has some drawbacks that should be taken into account:

- Limited content analysis: Content-based recommender techniques require sufficient information about the features or descriptions of items to accurately represent them. Therefore, the content of each item needs to possess a certain level of richness to enable precise user profile construction and subsequent recommendation.
- Overspecialization: Since content-based filtering suggests items similar to those previously rated, it may lack the ability to provide unexpected or novel recommendations. This phenomenon, known as the *serendipity problem*, restricts the system's capacity for generating highly diverse recommendations.
- New user challenge: To provide accurate recommendations, content-based filtering relies on collecting a sufficient number of ratings to better understand a user's interests. Consequently, the system may struggle to deliver reliable recommendations when the user has a limited number of ratings available.

Collaborative filtering makes predictions and recommendations by considering the rating data of other users within the system. It operates on the premise that if users exhibit similar rating patterns for certain items, they are likely to exhibit similar preferences for other items in the future. Among the various approaches to collaborative filtering, memory-based algorithms utilize the historical rating data to establish similarities between items or users. To facilitate this process, a user–item rating matrix is constructed, storing the ratings provided by users for the available items. Memory-based collaborative filtering can be further classified into two categories: user–user and item–item approaches.

- User–User Collaborative Filtering: This approach directly inherits the core definition of collaborative filtering. It involves constructing a neighborhood N of a target user by identifying users who exhibit similar rating behaviors. The ratings provided by these neighboring users are then used as a basis for generating predictions and recommendations for the target user.
- Item–Item Collaborative Filtering: Item–item collaborative filtering bears resemblance to early versions of content-based filtering in terms of its overall structure. However, in item–item collaborative filtering, the similarity between items is determined by their rating histories. Specifically, if two items are liked or disliked by the same set of users, they are considered to be similar. Similar to user–user CF, item–item collaborative filtering necessitates a similarity function to assess the similarity between two items. It also requires a method to generate predictions based on these similarities and ratings. In item–item collaborative filtering, a set S of items that have been previously rated by a user u and have the highest similarity to a given item i is collected to generate predictions.

Collaborative filtering offers several advantages which include the following [18]:

- Content independence: Collaborative filtering eliminates the need for content analysis, which implies that there is no requirement for the creation of language models, document analysis, or text preprocessing. This not only saves valuable time but also conserves valuable resources.
- Ease of user profile construction: Collaborative filtering simplifies the process of building user profiles. User profiles are created by compiling a list of items that the user has rated, along with their corresponding ratings. This approach avoids the complexity of weighing terms and determining their relevance in representing user interests.

Collaborative filtering suffers from several drawbacks including:

- Data sparsity: Users typically rate only a small fraction of the available items, resulting in a matrix with numerous missing values. This issue becomes more pronounced when attempting to recommend items to users with very few ratings.
- Cold Start Problem: Collaborative filtering faces a cold start problem when dealing with new users or new items. For new users who have not rated any items, the system struggles to find similar users for effective recommendations. Similarly, if a new item is introduced, the system lacks sufficient data to recommend it to any user.
- Limited scalability: As the number of users and items grows, collaborative filtering encounters scalability limitations. Storing user profiles and the user-rating matrix requires more memory. Additionally, the processing time increases significantly as more data needs to be analyzed to generate predictions for each user.

Hybrid recommender systems aim to enhance performance and leverage the strengths of different recommendation techniques by combining them [19]. Kim et al. [20] proposed three hybridization models: the linear combination model, the sequential combination model, and the mixed combination model. In the linear combination model, the output of multiple recommendation techniques is merged. Each technique generates a list of recommendations with assigned weights, which are then combined to produce a final recommendation list [11,21]. The sequential combination model first applies content-based filtering to construct user profiles and then utilizes collaborative filtering to generate recommendations. Content-based filtering can be used to either enrich the user's rating data [22,23] or construct user profiles considering the content of rated items [20]. The mixed combination model incorporates both semantic content and user ratings to generate recommendations. This approach combines the advantages of both techniques in the recommendation process. Apart from these hybridization strategies, other studies have combined multiple techniques using different approaches. In [24], each user has two user profiles constructed using content-based and collaborative filtering. Similarity between users is computed by combining the similarity scores using both user profiles using a weighted hybridization technique. Turnip et al. [10] uses content-based filtering on top of collaborative filtering to refine the list of recommendations such that only items with a certain degree of content similarity as the user's profile are retained. Logesh et al. [25] and Tran et al. [26] group users into clusters and employ user–user collaborative filtering within the clusters to predict missing ratings in the user–item rating matrix. Item–item collaborative filtering is then applied using the denser user-rating matrix.

1.2. Recommender system in opportunistic networks

Recommender systems designed for traditional environments face challenges when implemented in opportunistic networks, primarily due to the absence of a central entity. To overcome this limitation, the phases of recommendation need to be modified to accommodate the dynamic nature of the network. Existing literature on recommender systems in opportunistic networks distributes the responsibility of generating recommendations across all the participating nodes [27–29]. Each node possesses a local recommender system that handles the prediction/recommendation phase for its active user. The information collection phase is also implemented in a distributed manner where each node collects the necessary data needed by the local recommender system and stores it in a local database. Each node also handles the processing of the learning phase of the active user using the available data in the local database.

This study focuses on designing a hybrid recommender system specifically for a delay-tolerant digital marketplace environment. The dynamic nature of the network poses challenges in implementing a recommender system, as acquiring the necessary data for user profiling and generating predictions must be done opportunistically. Additionally, in the absence of a central entity, recommendation processing needs to be distributed across mobile participating nodes. Despite the limitations in communication network, the recommender system is expected to generate meaningful recommendations. To address this, a hybrid approach is employed, combining both content-based and collaborative filtering techniques. This hybridization aims to strike a balance between recommendation accuracy and coverage.

2. Materials and methods

Mobile devices with the necessary computing resources (e.g., CPU, bandwidth, and storage) are ubiquitous in the real world. We imagine these devices to cooperate and collectively form the communication infrastructure necessary for supporting a delay-tolerant digital marketplace. Using readily available network interfaces, each device defines a communication buffer. Whenever two devices come within the range of each other's buffer, a temporary link can be established, enabling bidirectional communication between them. This approach allows data transfers to occur even while the devices are mobile, as long as they remain within the limits of their communication buffers. However, it is important to note that the dynamic nature of the network, caused by node mobility, may result in frequent changes to the network topology. In order to extend the practical usability of this cooperative infrastructure beyond opportunistic encounters, each device is assumed to be equipped with a middleware application that implements a bundle protocol, such as IBR-DTN. This protocol enables the system to effectively handle disruptions and delays by storing message bundles locally until suitable intermediate nodes are encountered to assist in their incremental delivery.

A distributed clustering mechanism such as that described in [30] is assumed to be in effect for partitioning the network into smaller logical units that leverage node co-location for improving message delivery. The clustering process consists of two main phases: clustering setup and clustering maintenance. During the clustering setup phase, individual clusters and their respective cluster heads are identified using a highest degree heuristics approach. The node with the highest degree centrality, which represents the most connections, is chosen as the cluster head. In situations where multiple nodes have the same degree centrality, the tie is resolved by selecting the node with the smallest identifier. Once a cluster head is determined, the remaining nodes in the network affiliate themselves with a neighbor cluster head and become cluster members. If a node happens to be within the proximity of multiple cluster heads, it joins the bigger cluster. Clusters capitalize on the cluster head's high degree centrality to improve intra-cluster message delivery.

The network's dynamic topology may cause the cluster heads to be unable to reach cluster members. Similarly, it may encounter new nodes as it moves through space. To account for such situations, cluster maintenance is performed. Once a node disconnects from its cluster, it becomes a self-declared cluster head and signals its request to join a new cluster until it encounters a new cluster head. Encounters with non-cluster heads do not result in the disconnected node's joining of the cluster because the primary requirement of one-hop relationship with the cluster head is not satisfied.

The cluster head facilitates transfer of data only within its cluster. In order for nodes to transmit data to other clusters, another special node role is introduced. A gateway node is a member of a cluster that has at least one neighboring node from another cluster within its neighborhood set. These gateway nodes serve as important intermediaries, allowing clusters to establish connections and support inter-cluster communication. Cluster gateways maintain information about foreign neighbors in a gateway mapping table, which contains each foreign neighbor's unique ID and corresponding cluster ID. The gateway mapping table is regularly monitored and updated to ensure that the inter-cluster routes are active.

2.1. Hybrid recommender system

The hybrid recommender system proposed in this study leverages the concept of node clusters to facilitate message propagation throughout the network. While following the general phases of the recommendation process found in traditional online recommender systems (as discussed in Section 1.1.1), the proposed system incorporates certain modifications to accommodate the dynamic nature of the network.

Traditionally, the information collection phase in recommender systems focuses on gathering data to infer user interests. In the context of the proposed hybrid recommender system, this phase is extended to address the challenge of limited data accessibility in a dynamic network environment. The hybrid recommender system introduced in this study requires the collection of two types of data. First, historical rating data is utilized to implicitly capture and infer user interests. This data, generated by the user's previous ratings, serves as a basis

for understanding their preferences. Additionally, the system aims to obtain user profiles of individuals with similar interests to the active user. These profiles are acquired through opportunistic exchanges of user information with other nodes within the cluster. By sharing user profiles, the system gathers valuable data that fills the local database, which subsequently supports the operations of the local recommender.

The learning phase of the proposed hybrid recommender system operates in a distributed manner, with each node independently calculating the interests of its active user. The user's interest profile is represented as a vector of keywords extracted from the descriptions of items that the user has previously rated. This process does not rely on external data sources and can be carried out autonomously by each individual node.

During the prediction phase of the proposed hybrid recommender system, each node operates its own local recommender to generate item predictions for its active user. The local recommender relies on the data stored in its local database to make accurate rating predictions. This decentralized approach allows the recommender system to continue functioning even when nodes are disconnected from other nodes in the network.

2.1.1. Local database

The local database of the proposed hybrid recommender system consists of three buffers, namely: (1) *RatedProductsBuffer*, (2) *SimilarProfilesBuffer*, and (3) *ProductRecommendationsBuffer*.

The *RatedProductsBuffer* stores information about the items that have been previously rated by the target user. For each rated item, the buffer contains the product description and the corresponding rating provided by the user.

The *SimilarProfilesBuffer* is responsible for storing the user profiles obtained from opportunistic encounters with other nodes in the network. When a node receives a similar user profile, it compares the similarity between its own user profile and the received profile with the similarity with its current least similar neighbor (*lsn*). Initially, the *lsn* is set to −1. If the similarity between the node and the received user profile is higher than the current *lsn*, the least similar user profile in the *SimilarProfilesBuffer* is evicted to make space for the new profile. The *lsn* is updated to reflect this change and track the similarity of the least similar profile in the buffer.

The *ProductRecommendationsBuffer* contains the products that will be recommended to the target user. The specific algorithm for selecting and adding items to this buffer based on predicted ratings is beyond the scope of this study and is recommended for future development.

2.1.2. Interest extraction

Interest extraction in the proposed hybrid recommender system involves two phases: item profile creation and user profile creation. Each item i_j has an item profile which is a vector of keywords extracted from its item description. However, item descriptions often contain unstructured text, including characters or texts that are irrelevant and introduce noise to the item profile creation process. To address this, a text pre-processing procedure is applied to sanitize the item descriptions. The pre-processing procedure is illustrated in Fig. 3.

Using the sanitized item descriptions, item profiles are created using the algorithm used in [32]. Item profile $v(i_j)$ is denoted as,

$$v(i_j) = ((k^1_{i_j}, w(k^1_{i_j})), (k^2_{i_j}, w(k^2_{i_j})), \dots, (k^m_{i_j}, w(k^m_{i_j}))) \tag{1}$$

where $k^l_{i_j}$ corresponds to a keyword, $w(k^l_{i_j})$ denotes the weight of $k^l_{i_j}$, and m denotes the number of keywords in the description. The weight $w(k^l_{i_j})$ is computed using the following formula,

$$w_{k^l_{i_j}} = \frac{1 + log(n(k^l_{i_j}))}{\sum_{q=1}^{m}(1 + log(n(k^l_{i_j})))} \tag{2}$$

where $n(k^l_{i_j})$ corresponds to the number of occurrences $k^l_{i_j}$ has in the description.

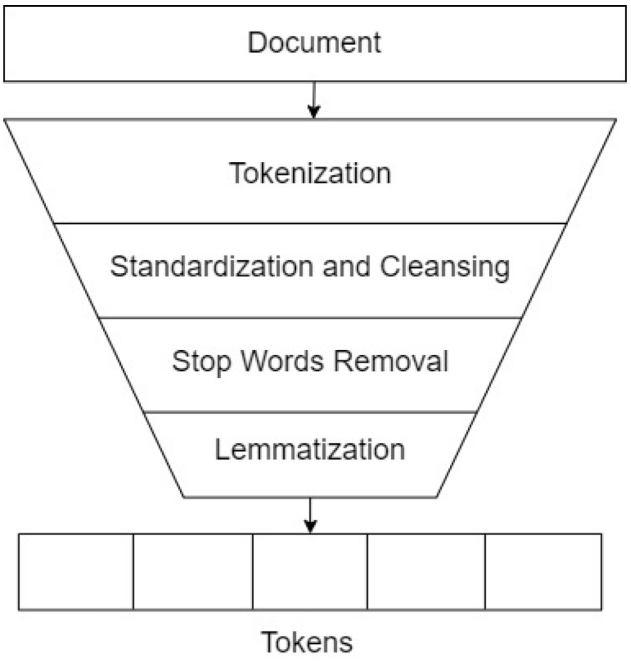


Fig. 3. Text pre-processing procedure taken from [31].

To construct the user profile, the item profiles of items previously rated by the user are utilized. Let $I_u = \{i^1_u, i^2_u, i^3_u, \dots, i^q_u\}$ be the set of items previously rated by user u. The user profile of u is denoted as follows:

$$v(u) = ((k^1_u, w(k^1_u)), (k^2_u, w(k^2_u)), \dots, (k^{m'}_u, w(k^{m'}_u))) \tag{3}$$

where k^o_u denotes a keyword, $w(k^o_u)$ denotes the weight of keyword k^o_u, and m' denotes the total number of unique keywords from all items in I_u. Instead of just computing the average weights of each keyword, the proposed approach also considers the actual ratings given to the items such that keywords found on highly-rated items will have higher weights. The weight $w(k^o_u)$ is computed as,

$$w(k^o_u) = \sum_{i=1}^{q} w(k_{u,i}) * (r_{u,i}) \tag{4}$$

where q denotes the total number of items in I_u and $r_{u,i}$ denotes the rating of user u on item i. The weight is further normalized using the following equation:

$$w(k^o_u) = \frac{w(k^o_u)}{\sum_{a=1}^{m'} w(k^a_u)} \tag{5}$$

2.1.3. Similar profiles propagation

The propagation of user profiles in the recommender system is highly dependent on the dynamic formation of clusters. Nodes within the system assume distinct responsibilities based on their specific roles in the cluster. The cluster head serves as a central entity in facilitating the dissemination of user profiles within the cluster. On the other hand, cluster members operate as clients, awaiting the cluster head's transmission of user profiles of their similar users within the cluster. This division of labor ensures efficient and targeted profile propagation, enhancing the overall effectiveness of the recommender system in capturing user preferences.

Each cluster head maintains a *MemberInformationTable* which records the following information for each of its cluster member.

- *lsn* — the similarity between the cluster member and the user profile of its least similar neighbor in the *SimilarProfilesBuffer*
- *user profile* — the user profile of the cluster member

- *pTime* — the time when the user profile was last updated locally by the cluster member.

Each cluster member actively communicates with its respective cluster head by sending a *BeaconMessage* to convey its current state. This message serves as a means of informing the cluster head of its presence and providing updates on its status. Upon receiving a *BeaconMessage* from a cluster member, the cluster head compares the *pTime* value in the message with the corresponding record in the *MemberInformationTable*. If the two values are not equal, indicating a change in the member's profile, the cluster head sends a *ProfileRequestMessage* to request the updated profile information from the cluster member. Subsequently, the cluster member responds by sending a *ProfileUpdateMessage* to the cluster head, ensuring that the profile information remains up to date within the cluster.

Upon receiving a *ProfileUpdateMessage* from a cluster member, the cluster head initiates an update procedure following the steps outlined in Algorithm 1. To optimize the update process and reduce the number of messages to be sent, a delayed broadcast mechanism is employed. When the cluster head receives a *ProfileUpdateMessage*, it starts an update timer. If subsequent *ProfileUpdateMessages* are received before the timer expires, the timeout period is reset. However, to prevent excessive waiting times, a maximum waiting time, denoted as *maxWaitTime*, is introduced. If the update timer reaches *maxWaitTime*, the update timer is no longer restarted, even if *ProfileUpdateMessages* are further received. This ensures that the system does not experience prolonged delays.

Algorithm 1 Update Procedure of Cluster Head

```
for each received user profile from a cluster member do
  addr = cluster member address
  newProfiles.add(addr, user profile)
  if updateTimer is running then
    updateTimer.reset();
  end if
  updateTimer.start()
end for
for each elapsed updateTimer do
  updateUserSimilarityMatrix(newProfiles);
  membersSimilarProfiles = gatherSimilarProfiles();
  sendSimilarProfiles(membersSimilarProfiles)
end for
```

After the update timer expires, the cluster head proceeds to update the user–user similarity matrix, which contains the similarity values between each cluster member. User profiles received during the update procedure are incorporated into the matrix. For each new profile, the cluster head computes the similarity with existing user profiles and updates the corresponding entries in the matrix. If a cluster member that provided a new profile already has an entry in the matrix, its entry is updated accordingly.

User–user similarity matrix is illustrated in Table 1. Each cell sim_{v_i,v_j} corresponds to the similarity between cluster members v_i and v_j. Similarity is calculated using cosine similarity, which determines similarity based on the angle between two vectors. The cosine-based similarity measure is computed as follows:

$$sim(v_1, v_2) = \frac{\sum_{a=1}^{m'} w(k^a_{v_1}) * w(k^a_{v_2})}{\sqrt{\sum_{a=1}^{m'}(w(k^a_{v_1}))^2} * \sqrt{\sum_{a=1}^{m'}(w(k^a_{v_2}))^2}} \tag{6}$$

where m' is the union of keywords from v_1 and v_2, and $w(k^a_{v_1})$ and $w(k^a_{v_2})$ corresponds to the weights of the ath keyword of m' in v_1 and v_2, respectively.

Once the matrix is updated, the cluster head proceeds to aggregate similar profiles for each member within its cluster. For every cluster member, the cluster head examines the user–user similarity matrix to identify other cluster members whose similarity exceeds the *lsn* value of the target cluster member. The user profiles of these identified cluster members are then aggregated and transmitted to the target cluster member via a *SimilarProfilesUpdateMessage*.

Upon receiving a *SimilarProfilesUpdateMessage*, the cluster member calculates the similarity between its user profile and each of the received user profiles from the cluster head. If the similarity is greater than its *lsn* value, the user profile is added to its *SimilarProfilesBuffer*, replacing the previously least similar user profile. On the other hand, if the received user profile does not meet the similarity criteria, it is disregarded.

Table 1
User–user similarity matrix.

	v_1	v_2	v_3	⋯	v_N
v_1	–	sim_{v_1,v_2}	sim_{v_1,v_3}	⋯	sim_{v_1,v_N}
v_2	–	–	sim_{v_2,v_3}	⋯	sim_{v_2,v_N}
v_3	–	–	–	⋯	sim_{v_3,v_N}
⋯	⋯	⋯	⋯	⋱	⋯
v_N	–	–	–	⋯	–

Table 2
User–user similarity matrix.

	u_s^1	u_s^2	u_s^3	⋯	u_s^P
u_{target}	sim_{u_{target},u_s^1}	sim_{u_{target},u_s^2}	sim_{u_{target},u_s^3}	⋯	sim_{u_{target},u_s^P}

2.1.4. Local recommender

Let u_{target} be the user profile of the target user and $U_s = u_s^1, u_s^2, u_s^3, \ldots, u_s^P$ be the set of user profiles in the *SimilarProfilesBuffer*. Each node keeps a user–user similarity matrix, which is a vector containing the similarity between u_{target} and the users in U_s. The representation for the user–user similarity matrix is shown in Table 2 where P corresponds to the number of users in U_s and sim_{u_{target},u_s^p} is the similarity between u_{target} and u_s^p.

Each node has a user–item rating matrix which contains the ratings of u_{target} and U_s. We define $I_R = i_1^R, i_2^R, i_3^R, \ldots, i_Q^R$ as the union of items rated by u_{target} and U_s. The user–item rating matrix can be represented as shown in Table 3, where Q represents the number of items in I_R and r_{u_p,i_q^R} signifies the rating given by user u_p for item i_q^R. The x-axis represents u_{target} and the users in U_s, while the y-axis represents the set of items in I_R. It is possible for the user–item rating matrix to have missing entries because u_{target} and U_s may not have the exact same set of previously-rated items.

To enhance the rating data of the target user, the local recommender employs collaborative filtering by predicting the missing ratings of u_{target} in the user–item rating matrix. For each unrated item i_q^R, the local recommender considers the ratings of similar users in U_s for that particular item using the following formula:

$$r_{u_{target},i_q^R} = \bar{r}_{u_{target}} + \frac{\sum_{p=1}^{P}(r_{u_p,i_q^R} - \bar{r}_{u_p}) * sim_{u_{target},u_p}}{\sum_{p=1}^{P}|sim_{u_{target},u_p}|} \tag{7}$$

where $\bar{r}_{u_{target}}$ and $\bar{r}_{u_p}$ denotes the average ratings of users u_{target} and u_p, respectively. r_{u_p,i_q^R} denotes the rating of user u_p on item i_q^R, sim_{u_{target},u_p} denotes the similarity between users u_{target} and u_p, P and Q corresponds to the number of users and items in the user-item matrix, respectively. To illustrate refer to Table 4. The entries for $r^R_{u_2,i_2}$, $r^R_{u_3,i_2}$, $r^R_{u_4,i_2}$, $r^R_{u_5,i_2}$ are considered when making rating predictions for $r^R_{u_{target},i_2}$ while $r^R_{u_3,i_2}$ is excluded because it has an empty value.

Let $I_N = i_1^N, i_2^N, i_3^N, \ldots, i_K^N$ be the set of items that needs rating predictions. An item–item similarity matrix is constructed which stores the similarities between items in I_R and I_N. Table 5 shows a representation

Table 3
User–item Rating Matrix.

	i_1^R	i_2^R	i_3^R	⋯	i_Q^R
u_{target}	$r^R_{u_{target},i_1}$	$r^R_{u_{target},i_2}$	$r^R_{u_{target},i_3}$	⋯	$r^R_{u_{target},i_Q}$
u_2	$r^R_{u_2,i_1}$	$r^R_{u_2,i_2}$	$r^R_{u_2,i_3}$	⋯	$r^R_{u_2,i_Q}$
u_3	$r^R_{u_3,i_1}$	$r^R_{u_3,i_2}$	$r^R_{u_3,i_3}$	⋯	$r^R_{u_3,i_Q}$
⋯	⋯	⋯	⋯	⋯	⋯
u_P	$r^R_{u_4,i_1}$	$r^R_{u_4,i_2}$	$r^R_{u_4,i_3}$	⋯	$r^R_{u_4,i_Q}$

Table 4
An example of a user–item rating matrix.

	i_1^R	i_2^R	i_3^R	i_4^R	i_5^R
u_{target}	5	?	3	4	
u_2	5	2	4		
u_3			3	4	4
u_4	4	3		4	4
u_5		2	3	4	

Table 5
Item–item similarity matrix.

	i_1^R	i_2^R	i_3^R	⋯	i_Q^R
i_1^N	$sim_{i_1^N,i_1^R}$	$sim_{i_1^N,i_2^R}$	$sim_{i_1^N,i_3^R}$	⋯	$sim_{i_1^N,i_Q^R}$
i_2^N	$sim_{i_2^N,i_1^R}$	$sim_{i_2^N,i_2^R}$	$sim_{i_2^N,i_3^R}$	⋯	$sim_{i_2^N,i_Q^R}$
i_3^N	$sim_{i_3^N,i_1^R}$	$sim_{i_3^N,i_2^R}$	$sim_{i_3^N,i_3^R}$	⋯	$sim_{i_3^N,i_Q^R}$
⋯	⋯	⋯	⋯	⋱	⋯
i_K^N	$sim_{i_K^N,i_1^R}$	$sim_{i_K^N,i_2^R}$	$sim_{i_K^N,i_3^R}$	⋯	$sim_{i_K^N,i_Q^R}$

of the Item–Item similarity matrix where $sim_{i_k^N,i_q^R}$ denotes the similarity between items i_k^N and i_q^R.

The local recommender checks the item–item similarity matrix to obtain the top l most similar items in I_R to i_k^N. To make rating prediction for item i_k^N, the ratings of u_{target} on the top l most similar items are considered using Eq. (8):

$$r_{u_{target},i_k^N} = \frac{\sum_{q=1}^{Q}(sim_{i_k^N,i_q^R} * r_{u_{target},i_q^R})}{\sum_{q=1}^{Q} sim_{i_k^N,i_q^R}} \tag{8}$$

where $sim_{i_k^N,i_q^R}$ denotes the similarity between items i_k^N and i_q^R and r_{u_{target},i_q^R} denotes the rating of u_{target} to i_q^R in the user–item rating matrix.

In general, when making predictions, the approach depends on whether the item in question has already been rated by the target user and its similar users, which is implied by an entry in the user–item rating matrix. If an entry exists, the predicted rating generated through collaborative filtering is utilized as the predicted rating for that item. On the other hand, if there is no existing entry for the item, the content-based filtering approach is employed instead. Therefore, the content-based filtering approach is only utilized when collaborative filtering is unable to make a rating prediction for the specific product.

2.2. Simulation environment

The ONE (Opportunistic Network Environment) simulator [33] is a simulation engine developed in Java that enables the execution of large-scale experiments in opportunistic networks. It provides the capability to assign diverse mobility models to different groups of nodes, allowing for the generation of movement patterns that mimic real-world agent navigation. Additionally, ONE allows users to specify the routing protocol to be employed by each node for message routing within the network. The interactions between nodes as they traverse the simulation environment can be visualized through ONE's graphical user interface or logged for future analysis. Leveraging ONE's application programming interface (API), we have developed a prototype delay-tolerant digital marketplace to evaluate the effectiveness of our proposed recommender system.

Table 6
Different groups of participating nodes in the simulation.

Group Name	Speed (m/s)	Mobility	Count
Pedestrians	0.5–1.5	SPMBM	129
Private vehicles	2.7–13.9	SPMBM	21
Public utility vehicles	2.7–13.9	MBM	20

2.2.1. Generating the simulation area

To generate road networks for simulation purposes, we begin by defining a bounding box to encompass the desired area of interest, which in this case is Tacloban City. The Overpass API provides a web interface that allows us to apply search filters, specifically `highway=*` and `type:way`, to extract only the relevant road networks. The extracted data is then exported in GeoJSON format. Next, we import the downloaded data into QGIS, an open-source Geographic Information System software, for further processing. Within QGIS, we utilize the *Disconnected Islands* plugin to group line segments into connected network groups. Among these groups, we select the one that represents the main roads, which will serve as the final road network for the simulation. The selected road network is exported in WKT format. The Coordinate Reference System (CRS) is set to EPSG:32651, specifically the WGS 84/UTM Zone 51 projection. These additional processing steps ensure the connectivity of all road segments within the simulation area.

2.2.2. Node groups and mobility models

In our experiments, we utilize groups to categorize and organize participating nodes based on their behavioral characteristics. We consider three distinct node groups: pedestrians, public utility vehicles, and private vehicles. Table 6 provides a summary of the key attributes associated with each group. Pedestrian nodes are designed to simulate human walking and jogging speeds, with movement velocities ranging from 0.5 m/s to 1.5 m/s. Meanwhile, vehicle-type nodes, representing public utility vehicles and private vehicles, exhibit speeds ranging from 2.7 m/s to 13.9 m/s. The upper speed limit corresponds to the widely accepted speed limits observed within towns across various countries. To drive node movement, we employ two mobility patterns: the map-based model (MBM) and the shortest-path map-based model (SPMBM). In the map-based movement approach, predefined routes are imported into the simulation environment, dictating the movement of nodes along specific paths. Conversely, in the shortest-path map-based model, nodes dynamically select random destinations and utilize Dijkstra's shortest path algorithm to determine the optimal routes to reach those destinations. Specifically, the shortest path map-based model is used by pedestrians and private vehicles, while public utility vehicles utilize the map-based mobility model to simulate their scheduled routes. The routes employed by the map-based mobility model are derived from real-world routes of public utility vehicles operating in Tacloban City, Philippines.

The total number of nodes that will be included in the experiments is set to 170 nodes. 20 nodes are allotted for public utility vehicles (PUVs) and the remaining 150 nodes are allotted for pedestrians and private cars. The distribution of the remaining nodes between pedestrians and private cars are based on available data on private vehicle ownership and the population of Tacloban City as indicated by the Land Transportation Office and the Philippines Statistics Authority.

2.2.3. Node initialization and simulation life cycle

In the simulation area, nodes are positioned based on their respective group names. Private vehicles and pedestrians are placed randomly on any point within the road network. On the other hand, public utility vehicles are constrained to initialize only at random points along the predefined routes imported into the simulation, simulating their scheduled paths. Predetermined randomization seeds are used for generating random calls during the simulation's life cycle. This guarantees

Table 7
General parameters used for recommender system simulation.

Parameter	Value
Simulation update interval	0.1 s
Scenario end time	9000 s
Request message size	1 KB
Default message size	300 KB
Message TTL	2 min
Routing protocol	Direct delivery
Transmission range	200 m
Message buffer size	50 MB
Item vector threshold	0.01
Item similarity threshold	0.3

consistency of events across different simulation scenarios and facilitates reproducibility. Once the simulation begins, nodes start moving according to their assigned mobility models. They establish connections with other nodes in close proximity, reflecting the opportunistic nature of the network. Throughout the simulation, reports are continuously updated to gather statistics that capture various aspects of the network's state, including encounters, topology, and buffer occupancy.

2.2.4. General simulation parameters

The general parameters and their default values are summarized in Table 7. The update interval of a simulation denotes the discrete time interval between event processing. It is believed that no events are processed between two timestamps. This parameter is set to 0.1 s to allow for the recording of events that occur during possibly brief contact between nodes. Simulation time for all scenarios is set to 9000 s. The transmission speed is set to 11 Mbps, which is equal to the IEEE 802.11b standard's theoretical maximum throughput. Messages received by each node are sent to adjacent neighbors using Direct Delivery as a routing protocol until they reach their destination or are dropped due to expired time-to-live values, which is set to 2 min. The parameters are configured in this manner because of the nature of our proposed approach which only sends messages to nodes that are in their respective transmission range. Therefore, the TTL of the messages only need to be long enough to be sent to the final recipient. When the messages are already received by their final recipient, they are deleted from the message buffer of the sender. At the same time, the messages are no longer added to the message buffer of the receiver. The message buffer size of each node is 50 MB and their corresponding transmission range is set to 200 m. The size of *RequestMessage* is 1 KB while the size of the other messages – *ProductMessage*, *SimilarProfilesMessage*, *ProfileUpdateMessage*, *BeaconMessage* – is set to 300 KB.

2.2.5. Dataset filtering

Users have different interests, which can be reflected on their purchase history. To emulate this, we use the user rating dataset extracted from Amazon website such that the nodes in the simulation will exhibit actual rating behaviors of real users in an e-commerce site. The dataset contains product metadata and reviews by actual users of the ecommerce platform from May 1996 to October 2018. Each product's metadata include its description, category, information, and prices, among others. There are 233.1 million total number of reviews and 29 product categories in the dataset. However, due to constraints in computing resources, only products under the *Grocery and Gourmet* category are considered which contains 283,354 products, 4,887,517 ratings by 2,695,230 users. The dataset is further filtered to include only users with at least 20 ratings to make user profiles more representative.

Since the number of users in the reduced version of the dataset is 7689, exceeding the allotted number of nodes for the simulation, 170 users are randomly selected. The dataset contains the timestamp of the actual time the items were rated by the users in the website. Therefore, we split the ratings of each user into train and test set based on their chronological order. The train set is composed of the earlier ratings and the test set is composed of the last 5 ratings of each user. In this way, given the ratings of the users in the train set, we evaluate the performance of our proposed system in predicting the ratings of users on unknown items — in this case, the items in the test set.

2.3. Experimental setups

Offline experiments are performed to evaluate the performance of the proposed algorithm using a dataset which is static snapshot of the supplied ratings of users. The dataset is divided into train and test set where the former constitutes the profiles of the users and the latter is *held out* and is used to measure the accuracy of the predictions afterwards. Unlike other methods for evaluating recommender systems (such as online evaluation and user studies), offline experiments do not require interaction with actual users, allowing for a low-cost comparison of a wide range of algorithms.

2.3.1. Metric

A good recommender system should be able to make predictions that are as close to the actual ratings of the user as possible. One of the most commonly used metrics to evaluate recommender systems is the Mean Absolute Error (MAE). MAE measures how deviated the predicted ratings are to the actual ratings of the user. Therefore, the lower the MAE is, the better is its performance in terms of prediction accuracy. MAE is computed as follows:

$$MAE = \frac{\sum_{(u,i)\in T} |\bar{r}_{u,i} - r_{u,i}|}{|T|} \tag{9}$$

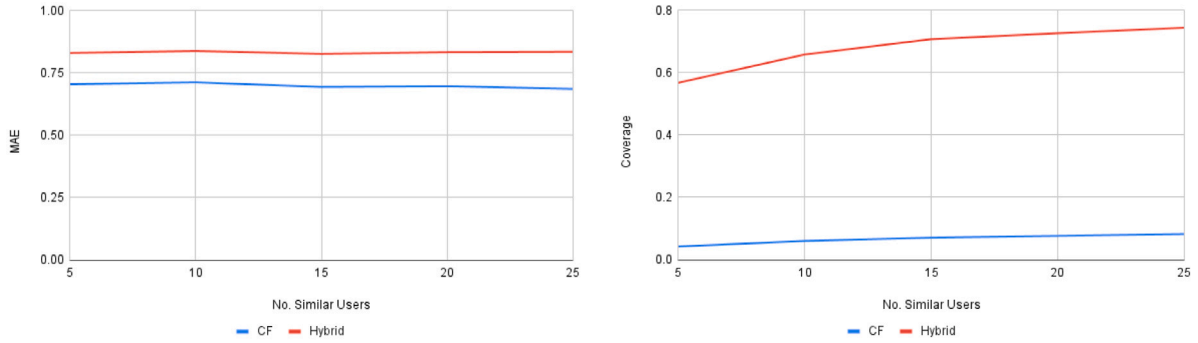
where T corresponds to the set of user–item pairs (u, i) in the test set, $\bar{r}_{u,i}$ and $r_{u,i}$ corresponds to user u's predicted and actual ratings to item i, respectively. If the recommender system fails to make rating predictions for a user–item pair, the pair is also disregarded even if it is included in the test set. Removing some entries that the recommender system cannot make rating predictions in the test set favors short-coverage systems. Therefore, a good recommender system should not only have high prediction accuracy, but high coverage as well. Coverage is defined as the proportion of available items that the recommender system can make rating predictions for. However, because of the limited resources, we cannot make rating predictions for all items available in the dataset. Instead, we only make rating predictions for each user–item rating pair in the test set. Thus, we calculate coverage as the percentage of user–item pairs in the test set that the system was able to make rating predictions for.

$$Coverage = \frac{|T_r|}{|T|} \tag{10}$$

where T_r denotes the set of user–item pairs that the system was able to make rating predictions and T denotes the set of user–item pairs in the test set.

2.3.2. Baseline setup

To establish a baseline, the recommender system is tested in a controlled setup where users possess complete knowledge of all other users in the network. In this scenario, nodes are strategically positioned within communication range of each other, and their mobility is disabled. The objective of this experiment is to compare the performance of pure collaborative filtering as described in [29], hereinafter referred to as reference algorithm, with the proposed hybrid approach, utilizing the Amazon dataset. Note that a more recent work by Beierle et al. [27] uses pure collaborative filtering but varies the way at which profiles are propagated in the network. The experiment also aims to determine the maximum achievable performance of both approaches when nodes can access the user profiles of their most similar users in the network. To explore the impact of the number of similar users considered in rating predictions, the experiment varies the number of similar users from 5 to 25, with an interval of 5. This variation enables an analysis of how the MAE and coverage metrics change as the system incorporates the opinions of an increasing number of similar users for each target user.



(a) Mean Absolute Error (b) Coverage

Fig. 4. Performance of our proposed hybrid recommender system and pure collaborative filtering using Amazon dataset.

2.3.3. DTN set-up

The next set of experiments focus on how our proposed recommender system makes rating predictions on products in a delay-tolerant environment. Over time, the MAE and coverage may change as the nodes obtain the profiles of their most similar users. Therefore, we compute the MAE and coverage every 5.0 s from the start of the simulation to track the performance of the recommender system as it gathers the profiles of more similar users in the network. We compare the performance of the reference algorithm and our proposed hybrid recommender system in a DTN environment. This experiment determines the viability of our proposed hybrid approach in improving the performance of our reference algorithm in building a recommender system in a delay-tolerant environment. Finally, we conduct a comparison between the performance of the proposed hybrid approach in both a centralized setup and a DTN setup. This experiment aims to determine the degree of success achieved by the DTN version despite the challenging network conditions.

3. Results and discussions

Fig. 4 compares the performance of the reference algorithm and the proposed hybrid approach. The coverage of the proposed hybrid approach is higher than the coverage of the reference algorithm. Our proposed approach was able to perform well in terms of coverage despite the sparseness of the user–item rating matrix which signals that the proposed hybrid recommender system was able to alleviate the cold start problem for new items which is evident in this dataset.

The MAE of both approaches did not increase nor decrease as it considers the opinions of more similar users. Because the nodes in this part of the experiment are chosen at random, the nodes have low similarity with each other. There are instances where the nodes cannot fill up their corresponding *SimilarProfilesBuffer*. Therefore, their list of similar users considered in making rating predictions is no longer increased despite increasing the size of the *SimilarProfilesBuffer*.

The performance of the proposed hybrid approach and the reference algorithm are tested in a delay-tolerant environment where nodes no longer have global access to the profiles of all the users in the network. Instead, nodes obtain the profiles of similar users through opportunistic exchange over clustered topology. Both approaches are tested considering the ratings of different number of similar users in predicting the ratings of users on items. The results are shown in Figs. 5, 6, 7, 8, and 9.

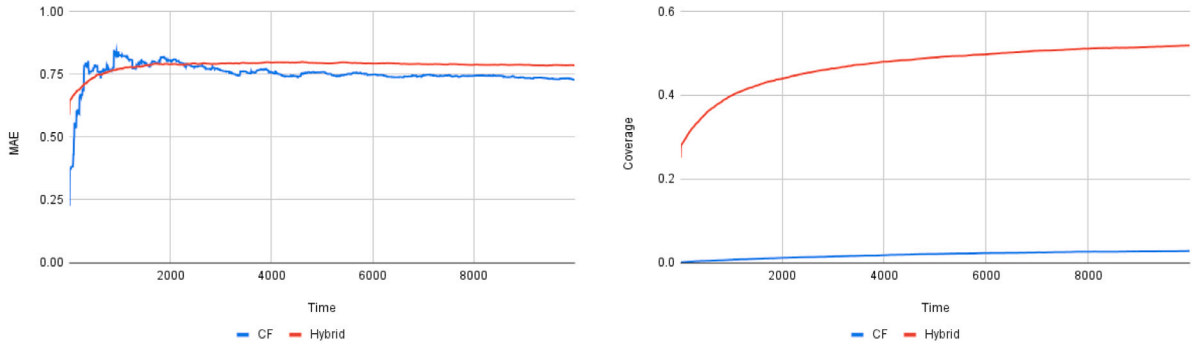
At the start of the simulation, the *SimilarProfilesBuffer* of all nodes are still empty because they have not had any contact with any other nodes in the network yet. The *SimilarProfilesBuffer* of each node is gradually filled over time. However, the similarity of the user profiles added to the buffer does not yet matter as long as the buffer is not yet full. This explains why the MAE of the reference algorithm is considerably higher at the start of the simulation. Meanwhile, for the hybrid approach, it does not purely rely on the ratings of its similar users but also on its own ratings. Therefore, the MAE of the proposed hybrid approach is not as high at the beginning as compared to the reference algorithm. The MAE of the reference algorithm becomes lower because the contents of the *SimilarProfilesBuffer* are eventually replaced with more similar user profiles as the simulation progresses. Meanwhile, the MAE of the proposed hybrid approach did not decrease but it remained consistent despite the increase in its coverage.

On the other hand, the results show that the coverage of both approaches increases as the recommender system considers the ratings of more similar users. The coverage of the reference algorithm initially starts at 0 which implies that it was not able to predict the rating of any product for any user. As mentioned previously, the *SimilarProfilesBuffer* of users at the start of the simulation is still empty; hence, the reference algorithm has no basis for making rating predictions at this time. Meanwhile, our proposed hybrid approach can already make rating predictions even if the *SimilarProfilesBuffer* of a node is still empty. However, at this point, the recommender system is reduced to pure content-based filtering because it will only rely on the ratings of the target user on items similar to the one that needs to be predicted. For both approaches, the coverage eventually increases as the recommender system considers the ratings of more similar users in the network.

Finally, the result of the proposed hybrid approach in the centralized set-up is compared with the final result of the DTN set-up. The result is shown in Fig. 10. For both approaches, the coverage of the centralized set-up is higher than the DTN set-up while the MAE is relatively similar. This entails that the centralized version performs better than the DTN set-up. However, the DTN set-up is not far off despite the challenging network conditions.

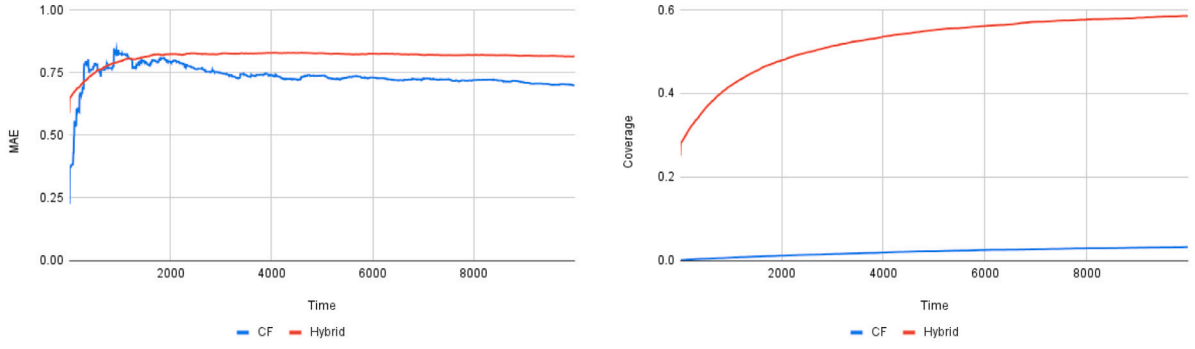
4. Conclusion

This study presents the development and evaluation of a recommender system designed for a delay-tolerant digital marketplace environment, utilizing a hybrid approach that combines collaborative filtering and content-based filtering techniques. Our proposed approach distributes the responsibility of the recommender system among nodes, enabling each node to generate rating predictions for its active user. To assess the performance of our proposed approach, we conducted simulations using The ONE framework. The results demonstrate that our hybrid approach can effectively make rating predictions even in the absence of a fixed network infrastructure. The baseline experiments



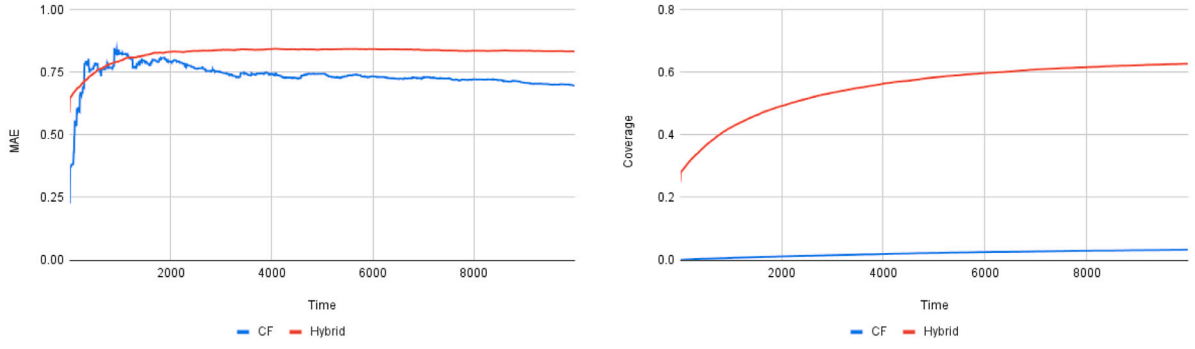
(a) Mean Absolute Error (b) Coverage

Fig. 5. Performance of proposed hybrid recommender system vs. pure collaborative filtering over delay-tolerant network considering the ratings of 5 similar users.



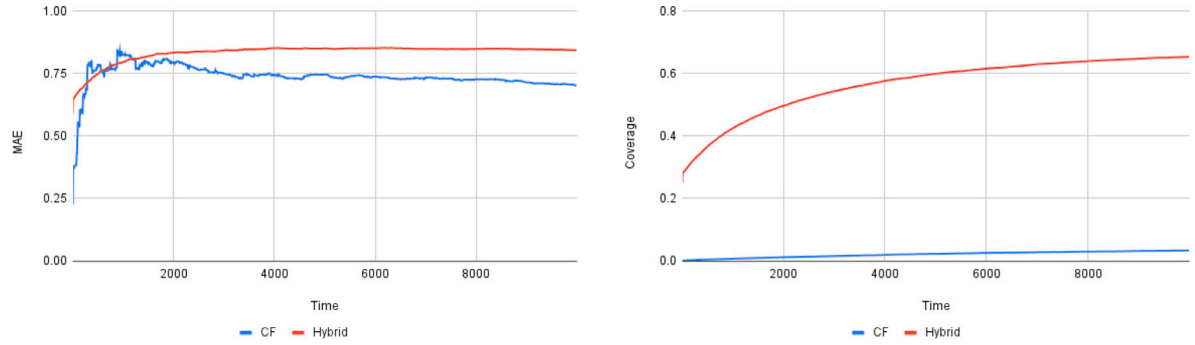
(a) Mean Absolute Error (b) Coverage

Fig. 6. Performance of proposed hybrid recommender system vs. pure collaborative filtering over delay-tolerant network considering the ratings of 10 similar users.



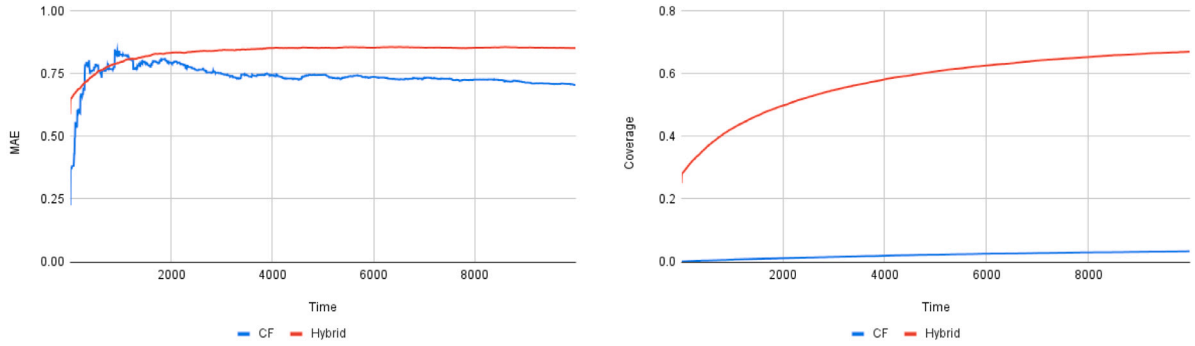
(a) Mean Absolute Error (b) Coverage

Fig. 7. Performance of proposed hybrid recommender system vs. pure collaborative filtering over delay-tolerant network considering the ratings of 15 similar users.



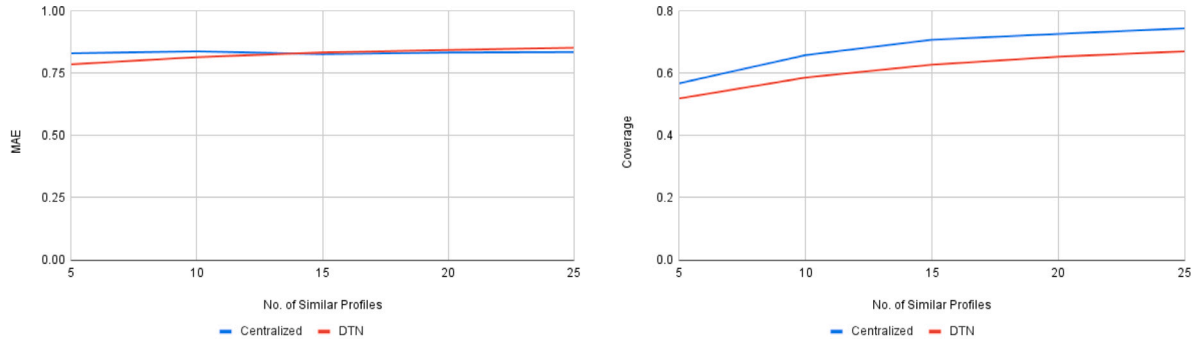
(a) Mean Absolute Error (b) Coverage

Fig. 8. Performance of proposed hybrid recommender system vs. pure collaborative filtering over delay-tolerant network considering the ratings of 20 similar users.



(a) Mean Absolute Error (b) Coverage

Fig. 9. Performance of proposed hybrid recommender system vs. pure collaborative filtering over delay-tolerant network considering the ratings of 25 similar users.



(a) Mean Absolute Error (b) Coverage

Fig. 10. Performance of proposed hybrid recommender system in centralized vs. DTN set-up.

revealed that the reference algorithm achieves better MAE, but at the expense of significantly lower coverage. In contrast, the proposed hybrid approach exhibits higher coverage while having a slightly higher MAE. The choice between these two approaches depends on the trade-off between higher coverage and a slightly higher MAE, or lower MAE with a smaller set of user–item pairs for rating predictions. Notably, the MAE of the proposed hybrid approach is expected to improve as more users provide ratings for items. In the delay-tolerant network environment, the proposed hybrid approach demonstrates its ability to make rating predictions even in the absence of user profiles from other network nodes. In such cases, the recommender system relies on pure content-based filtering. As nodes progressively obtain profiles of more similar users, the coverage of the system increases. This indicates that considering the ratings of users with higher similarity to the target user improves the coverage of the recommender system. Furthermore, the performance of the proposed hybrid approach in the DTN set-up is comparable to its performance in the centralized set-up, suggesting the feasibility of implementing the hybrid approach in a delay-tolerant environment.

CRediT authorship contribution statement

Victor M. Romero II: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Supervision, Project administration, Funding acquisition. **Bea D. Santiago:** Conceptualization, Methodology, Software, Validation, Formal analysis,

Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Jay Martin Z. Nuevo:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Visualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Victor M. Romero II reports financial support was provided by University of the Philippines Visayas - Office of the Vice Chancellor for Research and Extension.

Data availability

Data will be made available on request.

Acknowledgment

This study was supported by the University of the Philippines Visayas - Office of the Vice Chancellor for Research and Extension through its Regular In-house Research grant.

References

[1] Chenji H, Stoleru R. Delay-tolerant networks (DTNs) for emergency communications. In: Advances in delay-tolerant networks (DTNs). Elsevier; 2021, p. 103–34.
[2] Penning A, Baumgärtner L, Höchst J, Sterz A, Mezini M, Freisleben B. Dtn7: An open-source disruption-tolerant networking implementation of bundle protocol 7. In: Ad-hoc, mobile, and wireless networks: 18th International conference on ad-hoc networks and wireless, ADHOC-now 2019, Luxembourg, Luxembourg, October 1–3, 2019, Proceedings 18. Springer; 2019, p. 196–209.
[3] Baek KM, Seo DY, Chung YW. An improved opportunistic routing protocol based on context information of mobile nodes. Appl Sci 2018;8(8):1344.
[4] Burleigh S. Delay-tolerant electronic commerce. In: 2015 International conference on wireless communications & signal processing (WCSP). IEEE; 2015, p. 1–4.
[5] Schildt S, Morgenroth J, Pöttner W-B, Wolf L. Ibr-dtn: a lightweight, modular and highly portable bundle protocol implementation. Electronic Communications of the EASST 2011;37.
[6] Alamdari PM, Navimipour NJ, Hosseinzadeh M, Safaei AA, Darwesh A. A systematic study on the recommender systems in the E-commerce. IEEE Access 2020;8:115694–716.
[7] Galeano D, Paccanaro A. A recommender system approach for predicting drug side effects. In: 2018 International joint conference on neural networks (IJCNN). IEEE; 2018, p. 1–8.
[8] Berkani L. A semantic and social-based collaborative recommendation of friends in social networks. Softw - Pract Exp 2020;50(8):1498–519.
[9] Kbaier MEBH, Masri H, Krichen S. A personalized hybrid tourism recommender system. In: 2017 IEEE/ACS 14th International conference on computer systems and applications (AICCSA). IEEE; 2017, p. 244–50.
[10] Turnip R, Nurjanah D, Kusumo DS. Hybrid recommender system for learning material using content-based filtering and collaborative filtering with good learners' rating. In: 2017 IEEE conference on e-learning, e-management and e-services (IC3e). IEEE; 2017, p. 61–6.
[11] Wenzhen W. Personalized music recommendation algorithm based on hybrid collaborative filtering technology. In: 2019 International conference on smart grid and electrical automation (ICSGEA). IEEE; 2019, p. 280–3.
[12] Forouzandeh S, Aghdam AR, Barkhordari M, Fahimi SA, Vayqan MK, Forouzandeh S, Khani EG. Recommender system for users of internet of things (IOT). Int J Comput Sci Netw Secur 2017;17(8):46.
[13] Wang W, Duan L-Y, Jiang H, Jing P, Song X, Nie L. Market2Dish: health-aware food recommendation. ACM Trans Multimed Comput, Commun, Appl (TOMM) 2021;17(1):1–19.
[14] Sardianos C, Tsirakis N, Varlamis I. A survey on the scalability of recommender systems for social networks. In: Social networks science: Design, implementation, security, and challenges. Springer; 2018, p. 89–110.
[15] Sinha BB, Dhanalakshmi R. Evolution of recommender system over the time. Soft Comput 2019;23(23):12169–88.
[16] Lops P, Jannach D, Musto C, Bogers T, Koolen M. Trends in content-based recommendation: Preface to the special issue on Recommender systems based on rich item descriptions. User Model User-Adapt Interact 2019;29:239–49.
[17] Lops P, De Gemmis M, Semeraro G. Content-based recommender systems: State of the art and trends. Recomm Syst Handb 2011;73–105.
[18] Hameed MA, Al Jadaan O, Ramachandram S. Collaborative filtering based recommendation system: A survey. Int J Comput Sci Eng 2012;4(5):859.
[19] Çano E, Morisio M. Hybrid recommender systems: A systematic literature review. Intell Data Anal 2017;21(6):1487–524.
[20] Kim BM, Li Q, Park CS, Kim SG, Kim JY. A new approach for combining content-based and collaborative filters. J Intell Inf Syst 2006;27(1):79–91.
[21] Lenhart P, Herzog D. Combining content-based and collaborative filtering for personalized sports news recommendations. In: CBRecSys@ RecSys. 2016, p. 3–10.
[22] Geetha G, Safa M, Fancy C, Saranya D. A hybrid approach using collaborative filtering and content based filtering for recommender system. In: Journal of physics: Conference series, vol. 1000. IOP Publishing; 2018, 012101.
[23] Pal A, Parhi P, Aggarwal M. An improved content based collaborative filtering algorithm for movie recommendations. In: 2017 Tenth international conference on contemporary computing (IC3). 2017, p. 1–3. http://dx.doi.org/10.1109/IC3.2017.8284357.
[24] Li J, Xu W, Wan W, Sun J. Movie recommendation based on bridging movie feature and user interest. J Comput Sci 2018;26:128–34.
[25] Logesh R, Subramaniyaswamy V, Malathi D, Sivaramakrishnan N, Vijayakumar V. Enhancing recommendation stability of collaborative filtering recommender system through bio-inspired clustering ensemble method. Neural Comput Appl 2020;32:2141–64.
[26] Tran C, Kim J-Y, Shin W-Y, Kim S-W. Clustering-based collaborative filtering using an incentivized/penalized user model. IEEE Access 2019;7:62115–25. http://dx.doi.org/10.1109/ACCESS.2019.2914556.
[27] Beierle F, Eichinger T. Collaborating with users in proximity for decentralized mobile recommender systems. In: 2019 IEEE smartworld, ubiquitous intelligence & computing, advanced & trusted computing, scalable computing & communications, cloud & big data computing, internet of people and smart city innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI). IEEE; 2019, p. 1192–7.
[28] Eichinger T, Beierle F, Papke R, Rebscher L, Tran HC, Trzeciak M. On gossip-based information dissemination in pervasive recommender systems. In: Proceedings of the 13th ACM Conference on Recommender Systems. 2019, p. 442–6.
[29] Gratz P, Leclerc T. Delay-tolerant collaborative filtering. In: Proceedings of the 7th ACM international symposium on mobility management and wireless access. 2009, p. 109–13.
[30] Balaji R, Manokar V, Nandhini V, Shankar S, Vanitha S, Ali SB. A novel cluster based routing protocol in delay tolerant networks for consistency and energy efficiency. In: 2020 6th International conference on advanced computing and communication systems (ICACCS). IEEE; 2020, p. 1335–41.
[31] Anandarajan M, Hill C, Nolan T. Text preprocessing. In: Practical text analytics. Springer; 2019, p. 45–59.
[32] Chen K, Shen H, Zhang H. Leveraging social networks for P2P content-based file sharing in disconnected MANETs. IEEE Trans Mob Comput 2012;13(2):235–49.
[33] Keränen A, Ott J, Kärkkäinen T. The ONE Simulator for DTN Protocol Evaluation. In: SIMUTools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques. New York, NY, USA: ICST; 2009.