Transient Community Detection and Its Application to Data Forwarding in Delay Tolerant Networks

Xiaomei Zhang, Member, IEEE, and Guohong Cao, Fellow, IEEE

Abstract—Community detection has received considerable attention because of its applications to many practical problems in mobile networks. However, when considering temporal information associated with a community (i.e., transient community), most existing community detection methods fail due to their aggregation of contact information into a single weighted or unweighted network. In this paper, we propose a contact-burst-based clustering method to detect transient communities by exploiting pairwise contact processes. In this method, we formulate each pairwise contact process as a regular appearance of contact bursts, during which most contacts between the pair of nodes happen. Based on this formulation, we detect transient communities by clustering the pairs of nodes with similar contact bursts. Since it is difficult to collect global contact information at individual nodes, we further propose a distributed method to detect transient communities. In addition to transient community detection, we also propose a new data forwarding strategy for delay tolerant networks, in which transient communities serve as the data forwarding unit. Evaluation results show that our strategy can achieve a much higher data delivery ratio than traditional community-based strategies with comparable network overhead.

Index Terms—Transient community, data forwarding, delay tolerant networks.

I. INTRODUCTION

In Delay Tolerant Networks (DTNs) [1], mobile devices are only intermittently connected due to mobility and low node density. As a result, it is hard to maintain an end-to-end path, which makes data forwarding in DTNs extremely difficult. To address this problem, researchers have proposed approaches to exploit social network concepts, such as centrality [2]–[5], community [6]–[9], and friendship [10]–[12].

Communities have received considerable attention because of their applications to data forwarding in DTNs, worm containment [13], and other aspects. However, it is a challenge to detect communities in a large network [14] [15], especially considering the temporal information associated with communities. For example, a class community consisting of students attending a class may only appear during the day, and a dormitory community consisting of students in a dormitory may only appear at night. Since these communities normally appear during a time period and disappear thereafter, they are referred to as transient communities (TCs). Although many community detection methods have been proposed in the literature, there is hardly any method for TC detection.

Existing community detection methods are generally based on weighted networks or unweighted networks. For example, algorithms have been proposed in [16] and [17] to detect communities in weighted networks. Methods like label propagation [18] have been proposed to detect communities in unweighted networks. To detect communities in both weighted and unweighted networks, the Clique Percolation Method (CPM) [19], also known as K-clique, has been proposed. Recently, AFOCS [8] has been proposed to detect static communities and track community dynamics based on unweighted network snapshots. However, AFOCS aggregates contact information into a weighted or unweighted network. As a result, important contact information, such as the time when nodes contact, is lost. Losing such temporal information may result in two problems related to TC detection: false mixture and false separation, as shown in Figure 1 and Figure 2.

• False Mixture There are originally two TCs in the network, as shown in Figure 1 (a). TC1 is a class community that occurs during the day, and TC2 is a dormitory community that occurs at night. The two communities share several students, who take classes together and live together. Since there is a large overlap between them, traditional community detection methods may falsely mix these two communities as one. For example, CPM (K-clique) and AFOCS fail to distinguish the two communities when the overlap is larger than a certain threshold.

• False Separation Figure 2 (a) shows one TC in the network. Because the network is not strongly connected...
The contributions of this paper are summarized as follows: TC detection, we also apply TCs to data forwarding in DTNs. Since it is difficult to collect global contact information, we propose a distributed method. In addition to TC detection, we also apply TCs to data forwarding in DTNs. The contributions of this paper are summarized as follows:

- We propose a CCM method to detect TCs. Compared with existing methods, such as CPM and AFOCS, which do not consider the temporal information of communities, our method has much fewer false mixtures and false separations.
- Since it is difficult to collect global contact information, we further propose a distributed CCM method to detect TCs. Trace-driven simulation results show that the distributed CCM can effectively detect TCs.
- A TC may periodically appear, and therefore we propose techniques to identify this appearance pattern, which is useful in many applications.
- We propose a data forwarding strategy in DTNs based on TCs, where data are forwarded to TCs with better relaying capability to the destination, considering the time constraints of data. Evaluation results show that our strategy outperforms other existing data forwarding strategies in DTNs.

The rest of the paper is organized as follows. Section II presents our CCM method and distributed CCM method for TC detection. In Section III, we present the TC-based data forwarding strategy. Section IV provides an overview of the related work, and Section V concludes the paper.

II. TRANSIENT COMMUNITY DETECTION

In this section, we first give some preliminaries, and then we present our CCM method and the distributed CCM method for TC detection. Finally, we compare our TC detection method with other community detection methods and compare the distributed CCM method with CCM.

A. Preliminary

In this section, we describe the contact traces and introduce some definitions that will be used in this paper.

1) Contact Trace: We study the social contact patterns on three sets of traces: Dartmouth campus trace [20], MIT reality trace [21], and UCSD campus trace [22]. These traces record contacts among users carrying mobile devices on campus. In Dartmouth and UCSD traces, devices are WiFi-enabled. The Dartmouth trace uses SNMP logs from Access Points (APs). The original trace contains records of several thousands WiFi-enabled devices and lasts almost 4 years. In this paper, we focus on the data collected from September 2004 to December 2004. A contact is recorded when two devices detect the same AP simultaneously. The UCSD trace records the WiFi association of human-carried PDAs with APs, and a contact is recorded when two devices detect the same AP. In the MIT Reality trace, the devices periodically detect their peers via Bluetooth interfaces, and a contact is recorded when two devices move into the communication range of each other. The details of these three traces are shown in Table I.

2) Contact Burst: Simply extracting communities from weighted or unweighted networks is not enough to detect TCs. Therefore, instead of simplifying each pairwise contact process to a weight, our method processes the pairwise contact information directly.

From the trace summary, we can see that each trace has hundreds of thousands of contacts, and detecting TCs from these contacts directly will be difficult. Also, the opportunistic nature of the contacts hardly provides any clue on how they are related to TCs. Thus, we propose a different solution. We model the pairwise contact process using simple units, which can represent the contact information between nodes and be directly processed. We formulate each pairwise contact process as a series of contact bursts, during which most contacts between the pair of nodes appear. A contact burst is defined as follows:

Definition 1: A contact burst $T_B = [t_s, t_e]$ between two nodes is a time period when contacts frequently appear between these two nodes. Two adjacent contacts belong to one contact burst if and only if the inter-contact time between them is shorter than $\lambda$, where $\lambda$ is a pre-defined threshold.

Figure 3 shows three contact bursts of two nodes, where each vertical arrow indicates a contact and $T_{Bi}$ denotes the $i^{th}$ contact burst. A single contact is not considered a contact burst because it is more like a random contact.
With the concept of the contact burst, each pairwise contact process between two nodes is modeled as a series of contact bursts \( T_{Bi} = [t_{si}, t_{ei}], i = 1, 2, 3, \ldots \). To verify if the contact bursts can really represent node contacts, we have conducted some experiments based on the three traces to evaluate how many contacts can be represented by contact bursts. Table II shows the percentage of the contacts that are within the contact bursts and the duration of the contact bursts as a percentage of the total trace time. The value of \( \lambda \) is set empirically based on different traces. We have found that in all of our three traces, with \( \lambda = 1 \) hour, most contacts occur within some contact bursts, which only account for a small portion of the total time. We also evaluate how \( \lambda \) affects the percentage of the contacts that are within the contact bursts. The results are shown in Figure 4. As can be seen, there is a large percentage increase when \( \lambda \) increases from 0.5 to 1. As \( \lambda \) further increases, the percentage increase slows down. On the other hand, as \( \lambda \) increases, adjacent contacts with long inter-contact time are more likely to be falsely assigned to the same contact burst. Thus, we set \( \lambda = 1 \) hour in this paper.

Contact bursts are used to find TCs. A contact burst \( [t_{si}, t_{ei}] \) between a pair of nodes is the time interval in which the two nodes frequently contact. In a TC with duration \( [t_s, t_e] \), its members usually have frequent contacts with each other during this interval; i.e., their contact bursts are all with durations similar to \( [t_s, t_e] \). Therefore, if we are able to find a group of contact bursts with similar durations and they can mutually form a connected graph, they will form a TC. For example, in Figure 5, there are 10 contact bursts, where a double arrow represents a contact burst. Among them, six contact bursts start around 1 pm and end around 2 pm. Then, they can form a connected graph, and are most likely from the same TC with duration around \( [1, 2] \) pm. On the other hand, the other four contact bursts in the example have different contact durations, and are thus not in this TC. Although the similarity in this simple example can be easily calculated, in a complex network where contact bursts may not have the exact same starting and ending time, we need techniques to calculate their similarity.

### B. TC Detection With CCM

Based on contact bursts and their similarity, we cluster similar bursts together, from which we find TCs. There are many clustering methods in the literature, such as \( K \)-means [23], spectral clustering [24] and hierarchical clustering [25]. These methods aim to cluster a set of subjects, with the similarity well defined. Using the \( K \)-means algorithm, the number of clusters should be pre-known before clustering. Spectral clustering partitions nodes into clusters by using the eigenvectors of contact bursts.

### Table II

HOW CONTACT BURSTS CAN REPRESENT CONTACT PROCESSES (\( \lambda = 1 \))

<table>
<thead>
<tr>
<th>Trace</th>
<th>Percentage of the contacts within the contact bursts</th>
<th>Bursts’ duration as the percentage of total trace time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dartmouth</td>
<td>66.52%</td>
<td>6.05%</td>
</tr>
<tr>
<td>MIT</td>
<td>77.88%</td>
<td>10.36%</td>
</tr>
<tr>
<td>UCSD</td>
<td>81.22%</td>
<td>4.93%</td>
</tr>
</tbody>
</table>

### Figure 3

Three contact bursts. An arrow represents a contact between two nodes.

### Figure 4

How the percentage of the contacts within the contact bursts changes with parameter \( \lambda \).

### Figure 5

A connected graph built on contact bursts. Six contact bursts have similar contact durations, and they may be from one TC, circled by red line. The other four have different contact durations, which are not in this TC.
of a pairwise similarity matrix. Among them, hierarchical clustering is effective and efficient, with only one parameter—the minimum allowed similarity ($\gamma$) that can be flexibly set. Thus, our CCM is based on hierarchical clustering.

With a set of $n$ contact bursts, CCM runs as follows:

1) **Initialization**: Each of the $n$ contact bursts starts to form its own cluster.  

2) **Merge clusters**: Pick two clusters with the largest similarity and merge them together. The similarity of the two clusters is defined as the average of all the pairwise Jaccard similarity coefficients of the contact bursts in these two clusters. This step repeats until the termination condition is satisfied, as defined in the next step.  

3) **Termination**: The algorithm terminates if the largest similarity between all clusters in one round is smaller than the minimum allowed similarity $\gamma$.

In order for the algorithm to generate TCs, we need to modify the cluster merging phase. With this algorithm, two clusters with a similar occurring time may be merged as a TC. However, this is not always true. For example, if two clusters do not share any common node, it is more likely that they are from two TCs which happened at a similar time but at different locations. Thus, we add one condition for cluster merging; that is, the picked clusters must have at least one common node (as shown in Figure 6). For all pairs that satisfy this condition, those with the largest similarity are merged.

With CCM, contact bursts with similar time periods are clustered together. One cluster corresponds to one TC, where nodes attached to the contact bursts will be in that TC. When the algorithm terminates, there may be some “tiny” clusters, which only include one contact burst. One contact burst only consists of two nodes, and it is more like a personal meeting rather than a TC. For a contact burst $[t_s, t_e]$, its duration is defined as $t_e - t_s$. For a cluster (TC) that includes multiple contact bursts, its duration is defined as the average of the durations of these contact bursts, and its starting time and ending time are respectively defined to be the average of the starting time and ending time of the contact bursts. A TC with short duration may represent an occasional encountering of several nodes. Such TCs with short durations may not reappear and should not be counted as TCs. To further verify this, we conduct an experiment to test whether a TC’s reappearance is relevant to the TC’s duration. The result is shown in Figure 7. We find that in all three traces, the TCs with a very short duration (< 15 min) are less likely to reappear than those with a long duration ($\geq 15$ min). Therefore, we eliminate the TCs with durations shorter than a threshold $T_{min}$. We have tested different values for $T_{min}$ (i.e., from 10 min to 20 min), and find that there is no noticeable change on the reappearance ratio. Therefore, we simply set $T_{min} = 15$ min (0.25 hour). In addition, TCs with only one contact burst are also deleted, since these TCs only have two members. This is consistent with most existing work on community detection (e.g., [8], [19]), in which the detected communities should have at least three members.

**Parameter $\gamma$**: The minimum allowed similarity $\gamma$ affects the number of TCs and the community size. We run CCM on the UCSD trace to observe the impacts of $\gamma$, and the results are shown in Figure 8. As shown in the figure, increasing $\gamma$ results in more TCs with small community size. When $\gamma$ is small, the minimum allowed similarity between clusters becomes smaller. Then, more clusters are merged, resulting in fewer TCs, but larger community size. If $\gamma$ is too large, the clustering process ends quickly, and contact bursts that should belong to the same TC may not be clustered before the algorithm terminates. Thus, it is important to find the right value for $\gamma$.

In a mobile network, contacts that happen between members in the same community are called intra-community contacts, and contacts that happen between members in different communities are called inter-community contacts. A good community structure should have more intra-community contacts and fewer inter-community contacts. Since a TC also has temporal information, a contact in a TC is called an *intra-community contact* only when the contact happens between members in the same TC during the TC’s existing time period.

To obtain the percentage of intra-community contacts, we re-run the experiment with the TCs’ information and count the
number of intra-community contacts. The percentages of intra-
community contacts with different $\gamma$ are shown in Figure 9,
and a larger percentage is better. When $\gamma$ is small, more TCs
are merged even though their durations may be different. Then,
the durations of these individual TCs may be dramatically
different from the duration of the merged TC (the duration
of the merged TC is computed as the average of the durations
of these individual TCs). As a result, the occurring time of
most contacts may fall out of the duration of the merged TC
(i.e., the TC’s existing time period), and these contacts cannot
be counted as intra-community contacts. Therefore, when $\gamma$
is small, the percentage of intra-community contacts is low.
When $\gamma$ is large (e.g. close to 1), only TCs with very similar
durations are merged. Then, most of the resulting TCs have
very small sizes (less than 3), and these TCs will be eliminated
by CCM. Contacts in these eliminated TCs will no longer be
counted as intra-community contacts. Thus, the percentage of
intra-community contacts is also low when $\gamma$ is large. As can
be seen, the percentage of intra-community contacts reaches
maximum when $\gamma$ is 0.4, and we choose $\gamma = 0.4$ for the rest
of the paper. Note that many contacts are not intra-community
contacts even at $\gamma = 0.4$. This is because these contacts are
most likely random contacts, which are common in networks
with high node mobility.

C. TC Detection With Distributed CCM

CCM is a centralized detection method that requires all of
the pairwise contact information between nodes in the
network. Since it is difficult to collect global contact informa-
tion at individual nodes, we further propose a distributed
CCM. In this subsection, we first explain the basic idea of the
distributed CCM and then present the detailed algorithm.

1) Basic Idea: Similar to CCM, the distributed CCM detects
TCs by clustering the contact bursts. Since it is difficult
to collect all contact bursts at individual nodes, each node
only records the contact bursts with its neighbors, which are
referred to as local contact bursts. If a node $v$ has frequent
contacts with node $w$ in the recent time period and the adjacent
contacts have intervals smaller than $\lambda = 1$ hour, a contact
burst between $v$ and $w$ exists. An indicator $I_{(v,w)}(t) =
\{true, false\}$ is used to indicate whether the contact burst
between $v$ and $w$ exists at time $t$.

In addition to local contact bursts, each node also maintains
the TCs it has detected locally, which are referred to as
local TCs or local clusters (a TC is actually a cluster of
contact bursts). The local clusters detected at node $v$ should
include $v$ as a member and have pairwise similarities smaller
than the minimum allowed similarity $\gamma = 0.4$; otherwise,
similar clusters should be further merged. With this rule,
node $v$ adds new clusters to its local clusters in the following
two ways:

- **Adding local contact bursts:** As node $v$ finishes its contact
  burst with another node $w$ (i.e., the last contact between
  $v$ and $w$ happens $\lambda = 1$ hour before), the contact
  burst $T_{B}^{(v,w)}$ is treated as a new cluster $C_{new} = \{T_{B}^{(v,w)}\}$
  and is added to the local clusters of $v$. Since the new
  cluster may have a duration similar to existing local
  clusters (with a similarity larger than $\gamma$), the similar
  clusters should be merged.

- **Adding local clusters of other nodes:** Simply using the
  local contact bursts to detect TCs is not enough. Nodes
  also learn about the local clusters of other nodes upon
  pairwise contacts. Specifically, as node $v$ contacts node $w$,
  it exchanges the local clusters with node $w$. Afterwards,
  node $v$ selects some clusters from $w$ and merges with
  the local clusters at $v$.

2) The Distributed Algorithm: When a node $v$ initializes
the TC detection process, it has no local contact bursts and
local clusters. Whenever node $v$ encounters another node $w$,
when a node $v$ updates the local contact bursts and
exchanges the local clusters with $w$. Then node $v$ uses the new information to
update its local clusters. Algorithm 1 outlines the process
of updating the local clusters as node $v$ contacts node $w$ at
node $v$. We first give the notations used in the algorithm,
and then discuss the detailed process and the functions used in the algorithm.

At node $v$, $T_{B}^{(v,k)} = [\{I_{(v,k)}(t)\}, t_{e}^{(v,k)}]$ is used to denote the contact
burst with node $k$ ($k \neq v$), and $I_{e}^{(v,k)}(t) \in \{true, false\}$
indicates whether $T_{B}^{(v,k)}$ exists or not at time $t$. The set of local clusters
at $v$ is denoted as $C^{v}$. Each cluster $C$ in $C^{v}$ consists of a set of contact bursts. $S(C_{1}, C_{2})$ denotes the similarity
of two clusters. If $C_{1}$ and $C_{2}$ do not share any common node,
the similarity is 0; otherwise, it is computed as the average of all pairwise Jaccard similarity coefficients of the contact
bursts in the two clusters. $N$ denotes the set of nodes in the
network.

Algorithm 1 consists of two steps in general. In the first step
(Line 1-16), node $v$ updates the local contact bursts at time $t$.
Specifically, if $I_{e}^{(v,k)}(t) = true$ and the last contact in $T_{B}^{(v,k)}$
and its contact burst $T_{B}^{(v,w)}$ is extended to time $t$ by
including the current contact $(I_{e}^{(v,w)} \leftarrow t)$.
Algorithm 1 Updating the Local Clusters When Node \( v \) Contacts Node \( w \) at Time \( t \)
1: /* Update the contact bursts with other nodes. */
2: for \( k \in N \setminus v \) do
3: if \( I_{c}^{(v,k)}(t) = true \) and \( t - I_{c}^{(v,k)} \geq \lambda \) then
4: \( I_{c}^{(v,k)}(t) \leftarrow false; \)
5: \( C_{new} \leftarrow \{I_{c}^{(v,k)}\}; \)
6: \( C_{v} \leftarrow C_{v} \cup \{C_{new}\}; \)
7: NewCluster\( (C_{v}, C_{new}); \)
8: end if
9: end for
10: /* Update the contact burst between \( v \) and \( w \). */
11: if \( I_{c}^{(v,w)}(t) = false \) then
12: \( I_{c}^{(v,w)}(t) \leftarrow true; \)
13: \( t_{e}^{(v,w)} \leftarrow t, t_{c}^{(v,w)} \leftarrow t; \)
14: else
15: \( t_{c}^{(v,w)} \leftarrow t; \)
16: end if
17: /* Exchange the local clusters between \( v \) and \( w \). */
18: \( C_{v} \leftarrow \) the set of local clusters in node \( w \);
19: MergeClusters\( (C_{v}, C_{w}); \)

Function 1 - NewCluster
20: function NewCluster\( (C, C_{new}) \)
21: /* Find in \( C \) the most similar cluster \( C_{m} \) to \( C_{new} \). */
22: \( S_{m} \leftarrow 0; \)
23: for \( C \in C \setminus \{C_{new}\} \) do
24: if \( S(C, C_{new}) > S_{m} \) then
25: \( S_{m} \leftarrow S(C, C_{new}); \)
26: \( C_{m} \leftarrow C; \)
27: end if
28: end for
29: if \( S_{m} < \gamma \) then return
30: else
31: \( C_{m} \leftarrow C_{m} \cup C_{new}; \)
32: \( C \leftarrow C \setminus \{C_{new}\}; \)
33: NewCluster\( (C_{m}, C_{m}); \)
34: end if

Function 2 - MergeClusters
35: function MergeClusters\( (C_{1}, C_{2}) \)
36: \( S_{m} \leftarrow 0; \)
37: while \( S_{m} \geq \gamma \) do
38: /* Find the pair of cluster with highest similarity. */
39: \( S_{m} \leftarrow 0; \)
40: for \( C_{1} \in C_{1} \) do
41: for \( C_{2} \in C_{2} \) do
42: if \( S(C_{1}, C_{2}) > S_{m} \) then
43: \( S_{m} \leftarrow S(C_{1}, C_{2}); \)
44: \( C_{m} \leftarrow C_{2}; \)
45: end if
46: end for
47: end for
48: \( C_{new} \leftarrow C_{m}; \)
49: \( C_{1} \leftarrow C_{1} \cup \{C_{new}\}; \)
50: NewCluster\( (C_{1}, C_{new}); \)
51: end while

In the second step (Line 17-19), node \( v \) exchanges the local clusters with node \( w \) and checks if the local clusters of \( w \) can be merged with its local clusters. Node \( v \) uses the function MergeClusters to decide how to merge the clusters in node \( w \) with the local clusters. Generally, with MergeClusters, only clusters having high similarities with the local clusters are merged, since the clusters with low similarities usually have no relation to node \( v \).

Algorithm 1 only shows the updating process at node \( v \) as two nodes \( v \) and \( w \) contact. The updating process at node \( w \) is similar to the process at node \( v \) and therefore is not shown here. Next, we discuss the two functions used in Algorithm 1: NewCluster and MergeClusters.

- **NewCluster**: The objective of NewCluster is to check if the new cluster \( C_{new} \) should be merged with one of the existing clusters. It works by finding the cluster \( C_{m} \) that has the maximum similarity to \( C_{new} \). If the similarity is smaller than \( \gamma \), nothing needs to be done and the function returns. Otherwise, \( C_{new} \) is merged with \( C_{m} \) by adding the contact bursts in \( C_{new} \) to \( C_{m} \). \( C_{new} \) is then deleted from the local clusters. Since \( C_{m} \) has been updated by adding more contacts bursts, it is treated as another new cluster and the function NewCluster will be recursively called to check whether the updated cluster should be merged with other clusters.

- **MergeClusters**: Given two sets of local clusters \( C_{1} \) and \( C_{2} \), MergeClusters is used to select similar clusters in \( C_{2} \) and merge those clusters with the clusters in \( C_{1} \). The function works iteratively. In each iteration, the pairwise similarities between clusters in \( C_{1} \) and \( C_{2} \) are calculated. Then, for the pair of clusters that has the largest similarity, if the similarity is not smaller than \( \gamma \), the cluster from \( C_{2} \) is selected and added to \( C_{1} \) as a new cluster. Afterwards, the function NewCluster is used to check if the new cluster should be merged with the existing clusters in \( C_{1} \).

After all contacts have been processed, we use the same procedures as CCM to delete the clusters that only have one contact burst and delete clusters with a duration shorter than \( T_{min} \). Finally, each cluster corresponds to a TC which includes all nodes attached to the contact bursts.

D. Evaluations

1) Evaluations of CCM: We compare the performance of our TC detection algorithm CCM with two commonly used community detection algorithms: CPM (K-clique) and AFUCS. The performance is compared based on six metrics, covering various properties of community. Here, we only show the results based on the UCSD trace, since the results on other traces are similar. Based on the UCSD trace, the three algorithms are used to detect communities in each day over the trace’s period.

**Number of communities and community size**: Figure 10 (a) shows that CCM can detect many more TCs than communities detected by CPM and AFUCS. This indicates that CPM and AFUCS may have mixed some TCs into one community. Figure 10 (b) shows that communities detected by CPM are
usually bigger than those detected by AFOCS and CCM. This further confirms that CPM may have mixed some TCs, increasing the overall community size.

**Proportion of nodes involved in community:** Community structure is usually used to help data forwarding in DTNs; thus it is better if more nodes can be included in the community. Figure 10 (c) shows that CCM usually involves more nodes than CPM and AFOCS. In CPM and AFOCS, the node that has infrequent contacts with others may be ignored.

**Number of associated communities for one user:** We use this metric to show how much communities overlap. From Figure 10 (d), we can see that a node is normally attached to one community in CPM or AFOCS. This means communities are almost mutually disjointed, which does not achieve the objective of detecting overlapping communities. However, TCs detected in CCM have a strong overlapping property where a node belongs to an average of three TCs. Because TCs are detected using temporal information, it has no limitation on how many communities overlap. Other methods have limits on how much two communities can overlap.

**Proportion of intra-community contacts:** A community structure should incorporate more contacts. For CPM and AFOCS, a contact is considered an intra-community contact when it appears between two members in the same community. We need another temporal requirement for a contact in a TC to be considered an intra-community contact; that is, it must occur within the community’s duration. Therefore, we can see that TC incorporates less intra-community contacts than CPM and AFOCS, as shown in Figure 10 (e). CPM has the highest number of intra-community contacts. This can be explained by the fact that CPM detects many large communities, which counts more contacts as intra-community contacts.

**Location distortion:** It measures the difference between nodes’ locations within a community. Intuitively, a smaller location distortion means that users in the same community are near each other. On the contrary, a larger location distortion means that users may appear at multiple places in this community. A community’s location distortion is the standard deviation of the locations of all intra-community contacts. A contact’s location is calculated by averaging two node locations at the contact time.

\[
\text{Loc}_{<i,j>} = \frac{\text{Loc}_{i,t<\text{c}<j>} + \text{Loc}_{j,t<\text{c}<j>}}{2}
\]  

A community’s mean location is calculated by averaging the locations of all intra-community contacts.

\[
\text{Loc}(C) = \frac{\sum_{<i,j> \in C} \text{Loc}_{<i,j>}}{\sum_{<i,j> \in C} 1}
\]  

The **location distortion** within a community C is defined as the standard deviation of all intra-community contacts’ positions.

\[
\text{Distortion}(C) = \text{stdev}\{\text{Loc}_{<i,j>} \mid <i,j> \in C\}
= \sqrt{\text{var}\{\text{Loc}_{<i,j>} \mid <i,j> \in C\}}
= \sqrt{\sum_{<i,j> \in C} (\text{Loc}_{<i,j>} - \text{Loc}(C))^2} / \sum_{<i,j> \in C} 1
\]

The UCSD trace does not record users’ location information, so we estimate user location through the detected APs’ locations at that time. From Figure 10 (f), we can observe that the location distortion in TCs detected by CCM is smaller than that by CPM and AFOCS; i.e., nodes usually gather at one place in a TC. In a traditional community-based data forwarding strategy, data are intended to be forwarded to the communities that include the destination node. In such an application, a smaller location distortion is better, because this means nodes are near each other in one community. Once the data reaches the destination community, it must be near the destination node. In communities detected by CPM and AFOCS, nodes do not have a gathering period, so the contacts between them can appear at any time and any place, which results in a large location distortion.

**Verifying false mixture and false separation:** Here, we use examples to illustrate that traditional community detection method (i.e., AFOCS) may result in false mixture or false separation, while our CCM method works correctly.

**Example of false mixture:** To verify false mixture, we take one AFOCS community from one day of the UCSD trace as an example. The details of this community are shown in Table III. In Table III, **Member** lists the IDs of the community members. **Location** is the location of the community, as computed in Equation 2, represented as coordinates. **Location distortion...**
measures the difference between member locations within the community, as computed in Equation 3. As can be seen, the AFOCS community has six members: 3, 34, 48, 77, 101, 263, and a very large location distortion 181.537. The large location distortion indicates that the intra-community contacts between the six members occur at different locations. Then, we further detect transient communities during this day, and find that there are actually two TCs, as shown in Table IV. In Table IV, Duration is used to denote the existing time period of the TC, represented as the time in that day. As can be seen, TC1, which has five members: 3, 34, 77, 101, 263, occurred in the afternoon. TC2, which has three members: 48, 77, 101, occurred at night. The locations of the two TCs are far away, and the location distortion inside each TC is small. This indicates that TC1 and TC2 are from two distinct social activities that occurred at different time and locations. Because the two TCs share two common nodes, 77 and 101, they are falsely combined into one community by AFOCS, resulting in a false mixture.

Example of false separation: We also identify the existence of false separation in AFOCS. From one day in the UCSD trace, we identify two AFOCS communities, C1 and C2. As shown in Table V, C1 has four members: 112, 163, 225, 248, and C2 has five members: 16, 41, 180, 212, 225. C1 and C2 share one common node 225. The locations of the two communities are close to each other, and the location distortion inside each community is small. The location information indicates that members of C1 and C2 gather at the same place, and they may be from the same social community but are falsely separated. In fact, in the same day, we detect a large TC (as shown in Table VI) including all members of C1 and C2, and the location distortion of the large TC is small. This large TC can well represent the social community, in which all members of C1 and C2 participate. This also confirms that C1 and C2 are from the same TC which is falsely separated by AFOCS.

2) Evaluations of Distributed CCM: We next evaluate the performance of the distributed CCM based on the mobility traces. Effects of running time: Using distributed CCM, TCs are updated upon pairwise contacts. With a longer running time, each node encounters more nodes and more TC information can be learned. In the first experiment, we vary the running time and examine whether increasing the running time will enhance the performance of the distributed CCM. Here, the performance is evaluated based on the proportion of intra-community contacts. For distributed CCM, a contact is considered to be an intra-community contact if it is incorporated by the local TCs of the two nodes in contact. As can be seen from Figure 11 (a), in all three traces, increasing the running time from 1 day to 2 days enhances performance significantly, but further increasing the running time does not bring obvious performance gain. Therefore, we set the running time of the distributed CCM to be 2 days for the remainder of this paper. Comparing with CCM: We next compare the distributed CCM with CCM. The performance is first compared by measuring the proportion of intra-community contacts. As shown in Figure 11 (b), distributed CCM incorporate 80% – 90% as much contacts as CCM in all three traces. This demonstrates
that the TCs detected by distributed CCM (referred to as distributed TCs) can well represent the TCs detected by CCM (referred to as centralized TCs).

We further evaluate the distributed TCs by measuring their similarities to the centralized TCs. To measure the similarity between two communities, previous work [26] calculates the Jaccard similarity coefficient between members in the two communities. Since TCs also consider the duration of the communities, the similarity on duration should also be considered. Similarly, the similarity on durations is calculated using the Jaccard similarity coefficient. The similarity between two TCs $TC_1$ and $TC_2$ is calculated as the average of similarities on members and durations:

$$S_{m,t}(TC_1, TC_2) = \frac{S_m(TC_1, TC_2) + S_t(TC_1, TC_2)}{2}$$

where $S_m(TC_1, TC_2)$ and $S_t(TC_1, TC_2)$ respectively denote the similarities on members and durations.

In order to measure the similarity of a distributed TC $TC_d$ to the centralized TCs, we find the corresponding centralized TC $TC_c$ that has the highest similarity to $TC_d$. It is preferable if every distributed TC can find a corresponding centralized TC with high similarity. Otherwise, it usually means the detected distributed TC is inaccurate or unnecessary.

The distribution (CDF) of the similarities of distributed TCs to centralized TCs is displayed in Figure 11 (c). For the MIT and Dartmouth traces, more than 80% distributed TCs can find corresponding centralized TCs with similarity larger than 0.7. For the UCSD trace, about 50% distributed TCs can find corresponding centralized TCs with similarities larger than 0.7. Thus, most of the detected distributed TCs can find corresponding centralized TCs with high similarity. In the meantime, for all three traces, there exist some distributed TCs with very low similarities to the centralized TCs, which implies that the distributed CCM may detect some TCs that are inaccurate or unnecessary.

III. APPLICATION TO DATA FORWARDING

Community has been widely used for data forwarding in DTNs. However, ignoring community appearance time may lead to non-optimal forwarding paths. For example, in community-based data forwarding, data are always intended to be forwarded to a node within the destination’s community. This is not optimal when considering two problems. First, because a traditional community usually has a large location distortion, delivering data to the destination’s community does not mean it is getting closer to the destination node. Second, considering the appearance time of communities, some of the destination communities to which data are forwarded may not even appear before the data expire. Our TC-based data forwarding strategy solves these problems by utilizing TCs as the forwarding unit and always forwarding data to TCs that have better capability of relaying data to the destination node within a short time constraint.

Utilizing TCs to data forwarding requires knowledge on TC appearance patterns. In this section, we first study the periodic appearance patterns of TCs and discuss how to compute the relaying capabilities of TCs, and then present the TC-based data forwarding strategy.

A. Periodic Appearance of Transient Communities

With the proposed CCM, we can detect TCs on a daily basis. We run the algorithm based on the traces and find that there are many TCs that share similar members and appear at different times. These TCs represent a social group that appears periodically, like a class. In addition to these periodically appearing TCs, there exist TCs that only appear once. Such randomly appearing TCs are usually formed due to the opportunistic meeting of unfamiliar nodes. Later, we will no longer consider these opportunistically formed TCs. In this subsection, we will identify the periodic appearance of TCs based on the traces and exploit their appearance patterns.

1) Identifying the Periodic Appearances of TCs: The algorithm we use to cluster similar TCs is the same hierarchical clustering that we have presented in Section II-B. TCs are clustered according to the similarity of their members, which is also based on the Jaccard similarity coefficient.

We are interested in knowing how many times each TC appears. We run the clustering algorithm on three traces, each with 75 days of data. The CDFs of TCs’ appearance times are shown in Figure 12. TCs that only appear once are not included in the figure. As can be seen, more than 10% of TCs in the Dartmouth trace and about 5% of TCs in the MIT trace and UCSD trace appear more than 10 times. These TCs are considered as frequently appeared TCs, which may have the property of periodic appearance. Thus, the next question is how to find the appearance patterns of these frequently appeared TCs.

2) Appearance Patterns of TCs: In this subsection, we study the periodic appearance patterns of TCs on three sets of traces. We formulate the appearance patterns of TCs on a daily basis, based on the fact that many social groups, such as families, offices, and classes, are formed daily. Although there are many social groups formed on a weekly or monthly base or with a more complex pattern, we will not consider these patterns here and leave them as future work. The appearance pattern is formulated in two aspects. One is the distribution of the TC starting time, and the other is the distribution of TC duration.

We have two observations. First, the starting time of a TC within a day can be well approximated by a normal distribution. We take one TC in the Dartmouth trace as an example, and the distribution of the TC starting time is shown in Figure 13 (a). The parameters of the distribution are shown in Table VII. The TC usually appears at around 1 to 3 pm.
is that the TC starts in the time period Based on this, whether or not the TC appears within the pre-days when the TC has appeared during the warm-up period. The probability that a TC appears in a day is simply calculated as the percentage of and the distribution of its duration. The probability that a TC affected by three factors: the probability that the TC appears in determining the probability that a TC will appear. This is parameter exponential distribution, as shown in Figure 13 (b). The PDF of a TC’s starting time on a daily basis and the PDF of the TC’s duration. (a) Starting time. (b) Duration. Fig. 13. The PDF of a TC’s starting time on a daily basis and the PDF of the TC’s duration. (a) Starting time. (b) Duration.

Second, the duration of a TC can be approximated by an exponential distribution, as shown in Figure 13 (b). The parameter λ is 0.374 (as shown in Table VII), which means that the TC has an average duration of 1/λ = 2.674 hours.

In our traces, a TC only appears with limited times. As a result, the samples used to train the distributions are limited. Therefore, the approximation does not seem perfect, especially for the normal distribution. To further verify if the TC starting time follows normal distribution, we use Anderson-Darling Test [27] to test the normality of the distribution. The test reveals that about 72% of TCs pass the normality test for their starting time. This result is acceptable considering the limited sample size. If we only include the TCs that appear very frequently (e.g. appearing every day), the percentage of TCs that pass the test can reach 90%. If there are more data available, we believe that the distribution fitting would be better and more TCs can pass the normality test for their starting time.

Determining the probability of TC appearance within a time interval: Given a time interval \([t_s, t_e]\), we are interested in determining the probability that a TC will appear. This is affected by three factors: the probability that the TC appears in that day, the distribution of its starting time in that day, and the distribution of its duration. The probability that a TC appears in a day is simply calculated as the percentage of days when the TC has appeared during the warm-up period. Based on this, whether or not the TC appears within the predefined interval includes two possibilities. The first possibility is that the TC starts in the time period \([t_s, t_e]\), and the second is that the TC starts before the time interval but lasts until the start of the time interval. Suppose the starting time is represented by a normal distribution with parameters \(\mu\) and \(\sigma\), and the duration is represented by the exponential distribution with parameter \(\lambda\). The probability of the first possibility is:

\[
P_1[t_s, t_e] = \int_{t_s}^{t_e} \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(t-t_s)^2}{2\sigma^2}} dt
\]

where \(\text{normcdf}(t, \mu, \sigma^2)\) is the CDF of the normal distribution with mean \(\mu\) and standard deviation \(\sigma\). The probability of the second possibility is:

\[
P_2[t_s, t_e] = \int_{t_s}^{t_e} \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(t-t_s)^2}{2\sigma^2}} dt
\]

The part under the integral is actually the probability density function of the distribution \(\text{norm}(\mu + \lambda \sigma^2, \sigma^2)\). Thus, the integral can be easily computed by \(\text{normcdf}(T_1, \mu + \lambda \sigma^2, \sigma^2) - \text{normcdf}(0, \mu + \lambda \sigma^2, \sigma^2)\).

Then, the probability of the TC appearance within this time interval is:

\[
P_{TC}[t_s, t_e] = p_d \ast (P_1[t_s, t_e] + P_2[t_s, t_e])
\]  \hspace{1cm} (5)

where \(p_d\) is the probability that the TC appears in the day.

B. Relaying Capability

By estimating the current TC’s capability of forwarding data to the destination node within a future time period \([t, t + T]\), we can choose users in TCs with better forwarding capability as the data carriers. We denote the destination node as \(d\) and the current TC as \(TC_c\). The destination TCs that \(d\) belongs to are denoted as \(TC^d_1, ..., TC^d_N\). Specifically, we first compute the relaying capability of \(TC_c\) to each of \(TC^d_1, ..., TC^d_N\) within \([t, t + T]\), respectively. Then we obtain the relaying capability from \(TC_c\) to \(d\) by summing the computed relaying capabilities to \(TC^d_1, ..., TC^d_N\).

We next discuss in detail how to compute the relaying capability from \(TC_c\) to one of the destination TCs (\(TC^d_i\)). The relaying capability from \(TC_c\) to \(TC^d_i\) within the time period \([t, t + T]\) is computed by summing the probability that each node of \(TC_c\) will appear in \(TC^d_i\) in \([t, t + T]\). This is determined by the number of common nodes \(k\) between them and the probability that \(TC^d_i\) will appear in \([t, t + T]\). The probability that \(TC^d_i\) will appear in \([t, t + T]\), denoted as \(P_{TC^d_i}[t, t + T]\), is computed in Equation (5). Then, the relaying capability from \(TC_c\) to \(TC^d_i\) is computed as:

\[
R_{TC_c \rightarrow TC^d_i}[t,t + T] = k \ast P_{TC^d_i}[t,t + T]
\]  \hspace{1cm} (6)

The relaying capability from the current TC \(TC_c\) to the destination node \(d\) is computed by accumulating the relaying
capability from $TC_v$ to all the destination TCs $d$ belongs to:

$$R_{TC_v \rightarrow d}[t, t + T] = \sum_{i=1}^{n_d} R_{TC_v \rightarrow TC_i}[t, t + T]$$  \hspace{1cm} (7)$$

### C. TC-Based Data Forwarding Strategy

This paper focuses on unicast in DTNs, where the source node $s$, the destination node $d$, data’s initialization time $T_{in}$ and data’s expiration time $T_{ex}$ are given. The objective is to forward the data item from $s$ to $d$ within the time period $[T_{in}, T_{ex}]$. The basic idea of our TC-based data forwarding strategy is always forwarding data to a TC that has better relaying capability. Specifically, our strategy is based on the relaying capability of TCs in the recent time period $[t, t + T]$. Forwarding decisions are made upon node contacts. If a node meets another node that is in a TC with a larger relaying capability to the destination, data will be forwarded. Since the decision on data forwarding depends on the current TC the node belongs to, it is important for the node to know which TC it is currently in.

1) **Finding the Current TC:** To keep track of the TC that a node belongs to, each node keeps a queue ($Q_w$) of the recently met nodes and the contact time within the last time window ($T_w$). $T_w$ is a constant and how to set the value of $T_w$ will be discussed in Section III-E.2. Based on $Q_w$ and all the TCs that it has ever known, a matching score is assigned to each TC. The node will consider the TC with the largest matching score as the TC it currently belongs to. The matching score is computed as follows:

$$Score(TC) = \frac{\sum_{i \in Q_w \cap TC} (T_w - (t - t_i))}{|TC|}$$  \hspace{1cm} (8)$$

where $t$ is the current time, and $t_i$ is the node’s contact time with node $i$ in the queue.

Even though the distributed CCM method can also detect TCs distributedly, we do not use it to identify the current TC in the experiment. This is because distributed CCM cannot promptly detect TCs considering that the updates on contact bursts usually have delays. Therefore, we first use the CCM to detect TCs during the warm-up period. Then, during the experiment, we use the aforementioned method to quickly identify the current TC of each node by matching to the TCs detected in the warm-up period.

2) **Data Forwarding Based on TCs:** As TCs are the data forwarding units, data are always forwarded to the TC with better relaying capability to the destination. Therefore, once a data item reaches a new TC with a larger relaying capability, the data item is distributed to all nodes met in the TC. However, nodes do not carry the data permanently, and they only carry the data for a time period $T$. If the node that carries data no longer contacts any other node with a larger relaying capability or reaches a TC with larger relaying capability, it will delete the data at the end of $T$. The detailed data forwarding process is shown in the following steps:

When node $v_1$ contacts node $v_2$ at time $t$,

1) Update $Q_w$ for $v_1$, and find the TC that $v_1$ is currently in: $TC_1$.

2) For each data that $v_1$ carries.
   a) Check if the data should be deleted from $v_1$’s buffer. The data should be deleted when the node has neither contacted a node in a TC with a larger relaying capability nor gone to a TC with a larger relaying capability in the period $[t - T, t]$. Once the data item is deleted, check the next data item.
   b) Check if the data item is carried by $v_2$. If so, skip this data item.
   c) Check $v_2$’s current TC, $TC_2$. If $TC_2$ and $TC_1$ are the same, $v_1$ forwards data to $v_2$ so that the data can be distributed in the current TC. If $TC_2$ has a better relaying capability to $d$, $v_1$ also forwards data to $v_2$.

### D. Performance Evaluations

We evaluate the performance of the TC-based data forwarding strategy with the Dartmouth trace, the MIT Reality trace, and the UCSD trace, as presented in Section II-A.1. For each trace, we use the first half as the warm-up period, during which the TCs and TCs’ appearing patterns are detected and related information are transmitted among nodes. The second half of the trace is used to evaluate the performance. For each data item, the source and the destination are picked randomly and the generation time is randomly chosen in the daytime, since nodes’ activity remains low at night, which may result in in accurate comparison. Each experiment is repeated 1000 times for statistical convergence. The value of the time period $T$, with which we evaluate TCs’ relaying capability and decide when to delete data, is empirically set at each trace. We set $T = 1$ hour in all three traces, with which we can achieve high delivery ratio with a small number of data copies as can be seen in the following results.

Our TC-based data forwarding strategy is compared with three traditional community-based forwarding strategies and the Epidemic strategy that serves as the upper bound. A brief overview of these strategies is shown below:

- **Epidemic** [28]: When a node carrying the data item contacts another node, the data item is always forwarded to the contacted node if it does not have the data. This method has the best data delivery ratio and the highest network overhead. Therefore, it is used as an upper bound for comparison.

- **Label** [7]: This strategy is the first proposed community-based data forwarding strategy. In this strategy, the data item is forwarded to nodes that share at least one common community with the destination node. CPM (K-clique) is used to detect communities.

- **Bubble Rap** [6]: This strategy uses both centrality and community. CPM (K-clique) is used to detect communities. The data item is always forwarded to a higher centrality node, until it reaches a node that belongs to the same community as the destination node. When the data item reaches the destination community, it is forwarded to higher-centrality node within the community’s scope, until the destination node is reached.

- **AFOCS**: This strategy was proposed in [8], and it is used to evaluate the communities detected by the
AFOCS method. This method is based on how many common communities a node has with the destination node. The data item is only forwarded when the contacted node has more common communities with the destination node than the original carrier.

The performance is measured with three metrics, including the data delivery ratio, delay, and network overhead. The data delivery ratio is the proportion of data items successfully delivered before the data expires. The delay is the average time used to deliver each data item. The network overhead is the average number of data copies existing in the network at each moment. The results on the three metrics are respectively shown in Figure 14, Figure 15, and Figure 16. Generally speaking, our TC-based method performs much better than the other community-based methods by achieving a larger delivery ratio with less delay and comparable network overhead.

From Figure 14 we can see that, the TC-based strategy consistently performs better than the other community-based data forwarding strategies and even has comparable delivery ratio with Epidemic when the time constraint is small. The good performance within small time constraints is due to the fact that we study and predict users’ behavior in TCs within a short time period $T$ and forward data to nodes in TCs that have good relaying capability to the destination within $T$.

Figure 15 shows the delay of each strategy. As can be seen, the TC-based strategy consistently achieves low delay compared with other community-based strategies, demonstrating the superiority of the TC-based strategy on data forwarding.

Figure 16 shows the overhead generated by each strategy. Epidemic always has the highest overhead among all the strategies. Our TC-based strategy generates overhead comparable to Bubble Rap. Since the TC-based strategy removes data when there is no better TC formed or contacted within $T$. 

---

Fig. 14. Data delivery ratio of various methods based on three traces. (a) Dartmouth. (b) MIT. (c) UCSD.

Fig. 15. Delay of various methods based on three traces. (a) Dartmouth. (b) MIT. (c) UCSD.

Fig. 16. Data forwarding overhead measured by the number of data copies. (a) Dartmouth. (b) MIT. (c) UCSD.
the network overhead is kept low. Even though Label and AFOCS consume less network overhead than the TC-based strategy, the delivery ratio and delay of these two strategies are much worse than the TC-based strategy. Therefore, they cannot be effectively used for data forwarding in DTNs.

E. Discussions

1) Effect of Prediction on Performance: In our TC-based data forwarding strategy, predicting when and whether a TC will happen in the future is important. With prediction, we can compute the relaying capability of each TC to the destination node and then use the nodes in TCs with good relaying capability to carry data. To evaluate how the prediction affects the performance, we compare our strategy with a TC-based strategy without prediction (i.e., only distributing data when a node finds that the contacted node or itself is within the same TC of the destination node). Figure 17 compares their data delivery ratios. We can clearly see the superiority of the strategy with prediction.

2) Effect of Time Window $T_w$: In Section III-C.1, we used the information of recently met nodes in the last time window $T_w$ to identify the current TC. To find the optimal value of $T_w$ in each trace, we conduct an experiment to see how $T_w$ impacts the data delivery ratio. Here, the data delivery ratio is recorded with a time constraint of 12 hours. The results are shown in Table VIII. As can be seen, $T_w = 0.5$ hour has the best data delivery ratio for all traces.

3) Data Forwarding With Distributed CCM: For the current results on data forwarding, TCs are detected using centralized CCM. We further evaluate the performance of the TC-based strategy where TCs are detected using distributed CCM (referred to as distributed TC-based strategy). The strategy is compared with the strategy where TCs are detected using the centralized CCM (referred to as TC-based strategy). The results on the three traces are respectively shown in Figure 18, Figure 19 and Figure 20, where the performance is evaluated in terms of data delivery ratio and network overhead.

As shown in these figures, the distributed TC-based strategy has similar delivery ratio compared to the TC-based strategy in all three traces. This confirms our observation that the detected
distributed TCs can be used to represent most centralized TCs, including the destination TCs of each data item. By identifying the destination TCs, data can be successfully delivered to the destination. However, we find that the overhead of the distributed TC-based strategy is higher in most cases. This is because there are some unnecessary TCs detected by the distributed CCM, which create more data copies inside TCs.

IV. RELATED WORK

Complex networks usually consist of communities. Much research has been done on detecting communities by inter-disciplinary researchers from physics, biology, and computer science. They originally aimed to identify communities in a static network. There are many methods that focus on detecting disjoint communities, such as the modularity-based methods [16], [29] and label propagation [18]. Techniques have also been proposed to detect overlapping communities, such as CPM [19], link partitions [30], and AFOCS [8]. In addition, Domenico et al. [31] proposed a method to detect community structures in multilayer networks. A survey of existing community detection methods is given in [14]. The aforementioned community detection methods are all centralized methods, which require that the global network information is known a priori. Considering that global network information is hard to obtain in mobile networks such as DTNs, Hui et al. [26] proposed distributed community detection methods based on existing centralized methods. Clauset [32] proposed methods to detect local communities by distributedly exploring the graph near individual vertexes. In addition, Bae and Howe [33] achieved community detection in large-scale networks by proposing distributed detection algorithms.

Although community detection has been well studied in static networks, it remains difficult to detect time-varying communities in a dynamic network, such as a social network, in which people’s behaviors are highly dynamic. Recently, more research has been focused on studying evolving community structures based on network structures at successive network snapshots. Palla et al. [34] developed an algorithm to identify evolving communities by first detecting overlapping community structure at each network snapshot and then mapping similar communities at different snapshots. Nguyen et al. [8] proposed AFOCS, which can adaptively update the community structure at each network snapshot based on the history without detecting communities again at each snapshot. Similar methodologies are also utilized in [35]–[37] to detect evolving communities based on the networks at different snapshots. These methods are intended to analyze the long-term evolution of community structures caused by permanent change of humans’ behaviors or habits. There is still a dearth of work examining transient communities caused by the periodic changes of human behavior.

Although Gao et al. proposed the concept of transient community in [38], they only investigated the community relationship between individual pairs of nodes without providing complete knowledge about the transient community structure. Peixoto and Rosvall [39] studied dynamic community structure in temporal networks based on a Markov chain model. However, their focus was not DTN and they did not provide any approach to detect the reappearance of communities which is critical to data forwarding in DTNs. Pietiläinen and Diot [40] proposed algorithms to detect temporal communities that are similar to transient communities. The basic idea is to extract static community structure from each network snapshot. However, the time interval of the network snapshot is difficult to determine; thus it is challenging to accurately detect transient communities. Instead of detecting communities from network snapshots, our work directly studies nodes’ pairwise contact processes, and thus can accurately detect transient communities.

Community structure has been extensively utilized to address the problem of data forwarding in DTNs. It is believed that nodes within the same community have a higher chance of contacting each other. Hui et al. [6] and Hui and Crowcroft [7] have proposed strategies to forward data to the nodes that are within the destination community. Li et al. [41] and Wu et al. [42] defined the concepts of space-crossing communities and home communities, respectively, to better characterize human mobility and facilitate data forwarding in DTNs. However, these research efforts do not consider dynamic or temporal community structures. By introducing a dynamic community structure, AFOCS [8] further considered community evolution in community-based data forwarding. Transient communities in [38] were used to determine the scope for evaluating node centrality. Reference [40] studied how temporal communities contribute to data dissemination and concluded that temporal communities tend to limit data dissemination in DTNs. To the extent of our knowledge, our paper is the first work to fully utilize transient communities for data forwarding.

V. CONCLUSIONS

In this paper, we proposed a contact-burst-based clustering method (CCM) to detect TCs by exploiting the pairwise contact processes. We formulated each pairwise contact process as the regular appearances of contact bursts, during which most contacts between the pair of nodes appear. Based on this formulation, we detected transient communities by clustering the pairs of nodes with similar contact bursts. Trace-driven simulations showed that CCM can detect TCs more effectively compared with existing community detection methods. We also proposed a distributed CCM method to make the TC detection feasible in individual nodes, and demonstrated that this method can effectively detect TCs. Finally, TCs are applied to data forwarding in DTNs, where data are forwarded to TCs that have better relaying capability to the destination node. Trace-drive simulations showed that our strategy outperforms traditional community-based data forwarding strategies.

REFERENCES


Xiaomei Zhang received the B.E. degree from the University of Science and Technology of China in 2010, and the Ph.D. degree in computer science and engineering from The Pennsylvania State University in 2016. Since then, she has been with the Department of Mathematics and Computational Science, University of South Carolina Beaufort, where she is currently an Assistant Professor. Her research interests include mobile computing, mobile social networks, wireless communications, and data science.

Guohong Cao (S’98–A’99–M’03–SM’06–F’11) received the Ph.D. degree in computer science from Ohio State University in 1999. Since then, he has been with the Department of Computer Science and Engineering, The Pennsylvania State University, where he is currently a Professor. He has authored or coauthored over 200 papers in his research areas which have been cited over 17,000 times, with an h-index of 66. His research interests include wireless networks, mobile systems, Internet of Things, and wireless security and privacy. He has served on the Editorial Board of the IEEE TRANSACTIONS ON MOBILE COMPUTING, the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, and the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, and has served on the organizing and technical program committees of many conferences, including the TPC Chair/Co-Chair of the IEEE SRDS 2009, the MASS 2010, and the INFOCOM 2013. He was a recipient of the NSF CAREER Award in 2001.