Clustering and Cluster-Based Routing Protocol for Delay-Tolerant Mobile Networks

Ha Dang, *Member, IEEE,* and Hongyi Wu, *Member, IEEE*

***Abstract*—This research investigates distributed clustering scheme and proposes a cluster-based routing protocol for Delay-Tolerant Mobile Networks (DTMNs). The basic idea is to distributively group mobile nodes with similar mobility pattern into a cluster, which can then interchangeably share their resources (such as buffer space) for overhead reduction and load balancing, aiming to achieve efficient and scalable routing in DTMN. Due to the lack of continuous communications among mobile nodes and possible errors in the estimation of nodal contact probability, convergence and stability become major challenges in distributed clustering in DTMN. To this end, an exponentially weighted moving average (EWMA) scheme is employed for on-line updating nodal contact probability, with its mean proven to converge to the true contact probability. Based on nodal contact probabilities, a set of functions including *Sync()*, *Leave()*, and *Join()* are devised for cluster formation and gateway selection. Finally, the gateway nodes exchange network information and perform routing. Extensive simulations are carried out to evaluate the effectiveness and efficiency of the proposed cluster-based routing protocol. The simulation results show that it achieves higher delivery ratio and significantly lower overhead and end-to-end delay compared with its non-clustering counterpart.**

***Index Terms*—Clustering, delay-tolerant networks, routing.**

I. Introduction

As a natural consequence of intermittent connectivity among mobile nodes, especially under low nodal density and/or short radio transmission range, the Delay-Tolerant Network (DTN) technology [1], [2] has been introduced to mobile wireless communications, such as ZebraNet [3], Shared Wireless Info-Station (SWIM) [4], [5], Delay/Fault-Tolerant Mobile Sensor Network (DFT-MSN) [6]–[9], and mobile Internet and peer-to-peer mobile ad hoc networks [10]–[15]. DTN is fundamentally an opportunistic communication system, where communication links only exist temporarily, rendering it impossible to establish end-to-end connections for data delivery. In such networks, routing is largely based on nodal contact probabilities (or more sophisticated parameters based on nodal contact probabilities). The key design issue is how to efficiently maintain, update, and utilize such probabilities. Most DTN protocols [3]–[15] are "flat", where every node plays a similar role in routing. The flat architecture is simple and effective in small networks, but not scalable to large size DTNs.

Meanwhile, clustering has long been considered as an effective approach to reduce network overhead and improve scalability. Various clustering algorithms have been investigated in the context of mobile ad hoc networks. However, none of them can be applied directly to DTN, because they are designed for well-connected networks and require timely information sharing among nodes. A recent work [16] proposes a DTN hierarchical routing (DHR) protocol to improve routing scalability. DHR is based on a deterministic mobility model, where all nodes move according to strict, repetitive patterns, which are known by the routing and clustering algorithms. It cannot be generalized to such networks with unknown mobility as DTN-based peer-to-peer mobile ad hoc networks.

As discussed in [13], [17]–[19], a node in real-life tends to visit some locations more frequently than others. This issue is investigated in [18] by using real user traces obtained from Dartmouth College's campus. The study shows that students spend most of their school time at several specific locations on campus such as cafeteria, library, and study halls. It also shows that engineering students tend to stay in the engineering hall most of their time, while computer science students visit the computer science hall most often. A separate study [20] reveals that a real-life mobility pattern gives power-law-like time distribution on the visited locations. Subsequently, the authors of [19] introduce the *community-based* mobility model, where each node has its own home location that it visits most frequently, along with several elsewhere locations. Clearly, if two nodes share the same home location, they have high chance to meet each other. Thus real-life mobility patterns naturally group mobile devices into clusters.

In this work, we investigate distributed clustering and cluster-based routing protocols for Delay-Tolerant Mobile Networks (DTMNs). The basic idea is to autonomously learn unknown and possibly random mobility parameters and to group mobile nodes with similar mobility pattern into the same cluster. The nodes in a cluster can then interchangeably share their resources for overhead reduction and load balancing, aiming to achieve efficient and scalable routing in DTMN.

Clustering in DTMN is unique and non-trivial, because the network is not fully connected. Due to the lack of continuous communications, mobile nodes may have inconsistent information and therefore respond differently. As a result, it becomes challenging to acquire necessary information to form clusters and ensure their convergence and stability. Although it is largely understood by the research community that clustering helps to improve network scalability, no previous work has

Manuscript received September 9, 2008; revised January 23, 2009, September 15, 2009, and February 24, 2010; accepted March 12, 2010. The associate editor coordinating the review of this paper and approving it for publication was C. Xiao.

The authors are with the Center for Advanced Computer Studies (CACS), University of Louisiana at Lafayette, Lafayette, LA, 70504 USA (e-mail: {yxs4862, wu}@cacs.louisiana.edu).

This work is supported in part by National Science Foundation under Award Number CNS-0831823.

Digital Object Identifier 10.1109/TWC.2010.06.081216

been done in such emerging unique networks. Our research is the first effort to investigate the clustering problem and cluster-based routing in non-deterministic intermittent environment.

In our protocol, an exponentially weighted moving average (EWMA) scheme is employed for on-line updating nodal contact probability, with its mean proven to converge to the true contact probability. Subsequently, a set of functions including *Sync()*, *Leave()*, and *Join()* are devised to form clusters and select gateway nodes based on nodal contact probabilities. Finally, the gateway nodes exchange network information and perform routing. Extensive simulations are carried out to evaluate the efficiency of cluster-based routing. The results show that it achieves higher delivery ratio and significantly lower overhead and end-to-end delay, compared with its non-clustering counterpart.

The rest of the paper is organized as follows: Sec. II introduces the distributed clustering algorithm. Sec. III proposes cluster-based routing. Sec. IV presents simulation results and discussions. Finally, Sec. V concludes the paper.

II. Distributed Clustering

In this section, we introduce our proposed clustering algorithm for DTMN, which undergos the following steps. First, each node learns direct contact probabilities to other nodes. It is not necessary that a node stores contact information of all other nodes in network. Second, a node decides to join or leave a cluster based on its contact probabilities to other members of that cluster. Since our objective is to group all nodes with high pair-wise contact probabilities together, a node joins a cluster only if its pair-wise contact probabilities to all existing members are greater than a threshold γ. A node leaves the current cluster if its contact probabilities to some cluster members drop below γ. We will discuss more details about this threshold later in Sec. IV. Finally, once clusters are formed, gateway nodes are identified for inter-cluster communications. Two clusters communicate to each other mostly via gateways (although it is not mandatory, as to be discussed in Sec. III).

A. Challenges and Tactics

In an intermittent environment, end-to-end connections do not always exist. This uniqueness leads to several main challenges for clustering and routing as elaborated below:

a) Online Estimation of Contact Probabilities: Pair-wise contact probability has been widely used as a routing parameter in opportunistic networks. We also make use of it in our algorithm. However, one of the major problems in DTN is how to obtain this parameter distributively.

A naive approach is to keep the entire meeting history. This approach, while providing robustness, is costly in storage and lacks agility to adapt to changes in mobility pattern. Therefore we adopt a simple and effective approach, named exponentially weighted moving average (EWMA). More specifically, Node i maintains a list of contact probabilities ξ_{ij} for every other node j, which it has met before. ξ_{ij} is updated in every time slot, according to the following rule:

$$\xi_{ij} = \begin{cases} (1-\alpha)[\xi_{ij}] + \alpha, & i \ meets \ j \\ (1-\alpha)[\xi_{ij}], & otherwise, \end{cases} \quad (1)$$

where α is a constant parameter between 0 and 1, and $[\xi_{ij}]$ is the old contact probability.

Clearly, this is a dynamic process, and thus ξ_{ij} doesn't necessarily equal to the actual contact probability p_{ij}. However, our studies show EWMA effectively yields ξ_{ij}, whose mean converges to p_{ij}, which is important to ensure stable clusters. Formally, we establish the following theorem.

Theorem 1: If Nodes i and j have a probability of p_{ij} to meet in each time slot, EWMA yields ξ_{ij}, whose mean converges to p_{ij}.

Proof: Consider a sequence of time slots and let $\xi_{ij}(t)$ denote ξ_{ij} in time slot t. Clearly, the mean of $\xi_{ij}(1)$ is

$$E(\xi_{ij}(1)) = (1-\alpha)\xi_{ij}(0) + p_{ij}\alpha.$$

Similarly, we have

$$E(\xi_{ij}(2)) = (1-\alpha)^2\xi_{ij}(0) + p_{ij}\alpha[1+(1-\alpha)],$$

and

$$E(\xi_{ij}(t)) = (1-\alpha)^t\xi_{ij}(0) + p_{ij}\alpha[1+(1-\alpha)+\cdots+(1-\alpha)^{t-1}].$$

Let $t \to \infty$, we arrive at

$$\lim_{t\to\infty} E(\xi_{ij}(t)) = \alpha p_{ij}\frac{1}{\alpha} = p_{ij},$$

which depends on neither the parameter α nor the initial value $\xi_{ij}(0)$. ■

Although EWMA is proven to eventually arrive at the average which is equal to the nodal contact probability and the convergence is independent of α, it is in fact a random process that fluctuates around its mean value. Such fluctuation may lead to an estimation error of the nodal contact probability at a particular time. As a result, there is a trade-off in the selection of α. A smaller α results in less errors, but at the same time, longer delay to reach the steady state. In general, α should be chosen according to applications and application-specific requirements.

b) Fractional Clusters: Due to possible errors in the estimation of contact probabilities and unpredictable sequence of the meetings among mobile nodes, many unexpected small-size clusters may be formed. To deal with this problem, we employ a merging process that allows a node to join a "better" cluster, where the node has a higher stability as to be discussed in the next section. The merging process is effective to avoid fractional clusters.

c) Inconsistent Cluster Membership and Gateway Selection: The problem of inconsistency may appear in both cluster membership and gateway selection. For example, Node j leaves its current cluster C_q and joins the new cluster C_p. Given the network with low connectivity, other members of C_q are not timely informed of this change and thus falsely assume that Node j is still a cluster member. The inconsistency problem exists also in gateway selection for a similar reason. For instance, two nodes in the same cluster may have two different gateways to another cluster. Or a node may lose its gateway to an adjacent cluster because the gateway node has left. We deal with the inconsistency problems by employing a synchronization mechanism where nodes exchange and keep only the most up-to-date information.

d) Cluster Member with Low Contact Probability: A node with a very low nodal contact probability may still appear in the member list of another node. The main reason is that a mobile node may change its mobility pattern in real-life applications. For example, a student may have his/her regular mobility pattern in a semester (i.e., visiting certain class rooms, libraries, dormitories, cafeterias, etc.). When entering a new semester, his/her mobility may change due to the new/rescheduled classes and after-school activities. Similar scenarios will happen at holidays, summer/winter breaks, or when the student changes his/her major. When mobility pattern changes, but the member list is yet updated, the problem stated above may happen. A possible solution for this problem is to use timeout for membership binding. We will discuss this problem further in the next sections.

B. Clustering Meta-Information

Without loss of generality, a node, e.g., Node i, maintains its ID (i.e., i), its cluster ID (denoted by $\Omega(i)$), a cluster table, and a gateway table as its local information.

The cluster table consists of four fields, namely, Node ID, Contact Probability, Cluster ID, and Time Stamp. Each entry in the table is for a node ever met by Node i. For example, the entry for Node k contains its contact probability with Node i (denoted by ξ_{ik}) and its cluster ID (denoted by Ω_i^k). Note that both ξ_{ik} and Ω_i^k are updated according to the best knowledge of Node i, but not necessary to be always correct/accurate. For example, ξ_{ik} is generally unequal to p_{ik} (the actual contact probability between nodes i and k), although they are expected to be close, as discussed earlier. Similarly, Ω_i^k may not always be the correct cluster ID of Node k (i.e., $\Omega(k)$). The Time Stamp field (T_{ik}) contains the most recent clock time when Node i meets Node k. This field is used for synchronization as will be discussed later. The cluster members can be readily obtained from the cluster table. Let C_c^i denote the set of nodes in Cluster c, based on the information of Node i. Clearly, $C_c^i = \{k | \Omega_i^k = c\}$.

Node i maintains its gateway information in the gateway table, with four fields: Cluster ID, Gateway, Contact Probability, and Time Stamp. The Cluster ID field contains the list of clusters known by i. For each cluster, e.g., Cluster c, the Gateway field includes the ID of the gateway (denoted by G_i^c), while the Contact Probability field (ζ_i^c) indicates the highest contact probability between the gateway G_i^c and any node in Cluster c. Similarly, G_i^c and ζ_i^c are updated according to the best knowledge of Node i, and thus not necessary to be network-wide correct/accurate. Finally, the Time Stamp field contains the most recent time when the entry is updated.

C. Distributed Clustering Algorithm

The key part of the algorithm lies on the meeting event between any pair of nodes. A node then decides its actions subsequently. Specifically, a node will join a new cluster if it is qualified to be a member. Similarly, a node leaves its current cluster if it joins a new cluster, or it is no longer qualified to be in the current cluster. When two member nodes meet, they trigger the synchronization process to update their information. To this end, we define three main functions, namely Join, Leave, and Sync for the algorithm. During initialization, Node i creates a cluster that consists of itself only and two empty tables. Its cluster ID is set to be its node ID appended with a sequence number, i.e., $\Omega(i) =$*i:Seq*. Each node maintains its own sequence number, which increases by one whenever the node creates a new cluster, to avoid duplication. The algorithm is event-driven. Hereafter, Node i waits for three possible events, i.e., *Slot-Timeout*, *Meet-A-Node*, and *Gateway-Outdate*.

1) Slot-Timeout Event: Update Contact Probability: A *Slot-Timeout* event is generated by the end of every time slot, triggering the process of updating the contact probabilities by using the EWMA scheme discussed in Sec. II-A (i.e., Eq. 1).

Once the contact probabilities are updated, the *GatewayUpdate()* procedure is invoked to update the gateway table. As discussed earlier, the gateway table maintains a list of gateways to each cluster. Since Node i has updated its contact probabilities to all nodes, it may potentially choose better gateways. During this procedure, the following three cases are considered:

- Case 1: Consider an entry in the gateway table, e.g., for Cluster c. If Node i is not the gateway to Cluster c (i.e., $G_i^c \neq i$), it looks up the cluster table to identify Node k, to which it has the highest contact probability, among all nodes in Cluster c, i.e., $\xi_{ik} \geq \xi_{ik'}$, $\forall k, k' \in C_c^i$. If $\xi_{ik} > \zeta_i^c$ (the current gateway probability), the entry is updated, by setting $G_i^c = k$, $\zeta_i^c = \xi_{ik}$, and T_i^c to be the current clock time.
- Case 2: Still consider an entry for Cluster c in the gateway table of Node i. If Node i itself is the gateway (i.e., $G_i^c = i$). While Node i still identifies Node k with the highest contact probability as discussed above. If $\xi_{ik} \geq \hat{\gamma}$ (the gateway threshold), the entry is updated by setting $G_i^c = k$, $\zeta_i^c = \xi_{ik}$, and T_i^c to be the current clock time; otherwise $G_i^c = N/A$, $\zeta_i^c = 0$, and $T_i^c = N/A$ (or simply removes the entry). This is particularly important to keep the gateway information up-to-date.
  An example is shown in Fig. 1, where Nodes i and k were the gateways between their clusters before Node k joins another cluster, without notifying the change to Node i. If they meet again, Node i learns the change from Node k and thus updates its tables; otherwise, if they do not meet each other for a long time, ξ_{ik} becomes lower than $\hat{\gamma}$. In either case, Node i's gateway table should be updated correctly by using the above procedure.
- Case 3: Node i also checks the cluster table for possible new clusters not included in its current gateway table. If a new cluster is found, it will be added into the gateway table.

2) Meet-A-Node Event: Update Cluster Information: The *Meet-A-Node* event is generated upon receiving the Hello message (exchanged between two meeting nodes, i and j). A series of actions will be taken at both sides as elaborated below.

If Nodes i and j are in the same cluster, i.e., $\Omega(i) = \Omega(j)$, the membership check function is invoked to verify if they are still qualified to stay in the same cluster. Specifically, if $\xi_{ij} \geq \gamma$, they continue to stay in the same cluster, and perform the synchronization of their cluster information. The

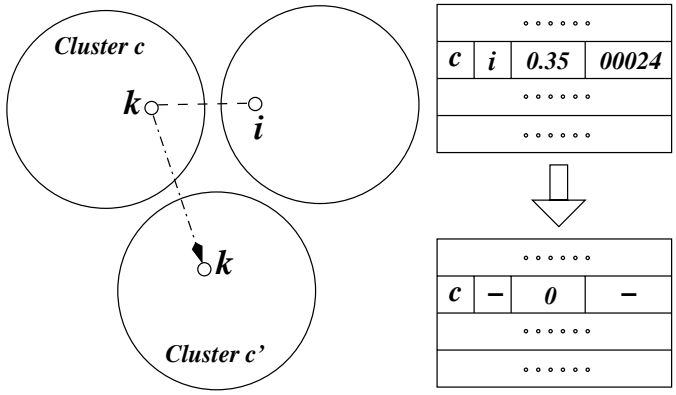
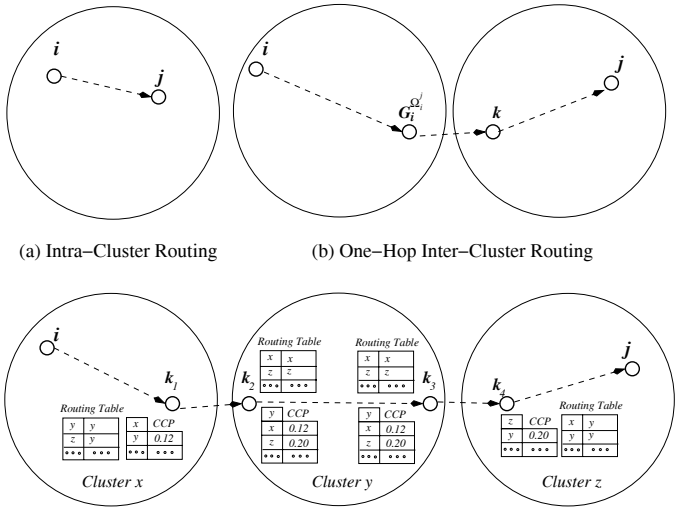


Fig. 1. Update of the gateway table.



(a) Intra–Cluster Routing (b) One–Hop Inter–Cluster Routing

(c) Multi–Hop Inter–Cluster Routing

Fig. 2. Cluster-based routing.

Sync() procedure includes two steps for synchronizing cluster members and gateways, respectively.

- *Synchronization of Cluster Members:* Note that, although $\Omega(i) = \Omega(j)$, $C^i_{\Omega(i)}$ and $C^j_{\Omega(j)}$ (the lists of cluster members at nodes i and j, respectively) are not necessarily identical, because the clustering is distributed and thus Nodes i and j may maintain different cluster information before the clustering procedure is converged. The basic idea of synchronization is to update the membership based on the latest information of Nodes i and j.
  More specifically, Node i sends to Node j a set of its current cluster members along with the time stamps, i.e., $\Psi = \{(k, T^k_i) | k \in C^i_{\Omega(i)}\}$. Upon receiving Ψ, Node j divides it into two subsets based on time stamps, $\Psi_1 = \{(k, T^k_i) | k \in C^i_{\Omega(i)}, k \notin C^j_{\Omega(j)}, T^k_i > T^k_j\}$ and $\Psi_2 = \{(k, T^k_i) | k \in C^i_{\Omega(i)}, k \notin C^j_{\Omega(j)}, T^k_i < T^k_j\}$. The former is the set that Node i has newer information (as $T^k_i > T^k_j$), while Node j's information on the latter is newer. As a result, Node j updates its cluster table according to Ψ_1, by setting $\Omega^k_j = \Omega(i) = \Omega(j)$, $\forall k \in \Psi_1$.
  Meanwhile, it sends $\{(k, \Omega^k_j) | k \in \Psi_2\}$ to Node i, which in turn updates its cluster table by setting $\Omega^k_i = \Omega^k_j$, $\forall k \in \Psi_2$. Node j in turn sends its current cluster members to Node i for a similar process.
- *Synchronization of Gateways:* Nodes i and j may have different gateways to the same cluster(s) in their gateway tables. Thus synchronization is needed to keep the "better" one with higher contact probability. For each of such clusters, the node whichever has lower contact probability gives up, and updates its gateway table. For example, consider Cluster c and assume $\zeta^c_i < \zeta^c_j$. Then, Node i updates its gateway table by setting $G^c_i = G^c_j$ and $\zeta^c_i = \zeta^c_j$. However, the Time Stamp, T^c_i, is *not* updated, unless Node j is the gateway itself.

If Nodes i and j don't pass the membership check, one of them must leave the cluster. Two issues are involved in the *Leave()* procedure. First, we identify the leaving node based on its stability, defined as its minimum contact probability with its cluster members, i.e., $min\{\xi_{ik} | k \in C_{\Omega(i)}\}$ for Node i. The node with lower stability leaves. It indicates the likelihood that the node will be excluded from the cluster due to its low contact probability. If there is a tie, the node with higher ID is chosen. Second, the leaving node creates a new cluster that consists of itself only. It keeps the current cluster table, because all information in the table is still valid, and resets the gateway table to be empty. Then, it sends its new cluster ID to the other node to update the cluster table accordingly.

If $\Omega(i) \neq \Omega(j)$, Nodes i and j consider whether or not to join each other's cluster. The *Join()* procedure is employed for a node to join a better cluster or to merge two separate clusters. It includes three steps. First, they exchange their member list (i.e., $C^i_{\Omega(i)}$ and $C^j_{\Omega(j)}$) and perform membership check. For example, Node i calculates ξ_{ik}, $\forall k \in C^j_{\Omega(j)}$. If any ξ_{ik} is less than γ, the membership check procedure returns false. If none of them pass the membership check, no further action is taken. If a node passes, it goes ahead to perform *Join()* function. If both nodes pass, the one with lower stability joins the other cluster. Second, the selected node updates its cluster ID and cluster table. For example, assume Node i joins Node j's cluster. It sets $\Omega(i) = \Omega(j)$, and $\Omega^k_i = \Omega(j)$, $\forall k \in C^j_{\Omega(j)}$. Third, it copies the gateway table from Node j.

3) Gateway-Outdate Event: Reset Gateway: When the Time Stamp of any entry in the gateway table is older than a threshold, Δ, a Gateway-Outdate Event is generated for that entry. For example, if $t - T^c_i > \Delta$ where t is the current clock time, the gateway G^c_i is likely lost. Thus the gateway table entry for Cluster c is reset, by letting $G^c_i = N/A$, $\zeta^c_i = 0$, and $T^c_i = N/A$.

Based on the above methods for updating contact probabilities, cluster and gateway tables, the proposed distributed clustering protocol always arrives at fast converged and stable clusters that comprise nodes with similar mobility patterns, as demonstrated in our extensive simulations.

III. Cluster-Based Routing

Once the clustering procedure is finished, each node in the network is associated with a cluster. For any two clusters whose members have high enough contact probability ($\geq \hat{\gamma}$), a pair of gateway nodes are identified to bridge them. In this section, we discuss how to route data messages efficiently in DTN, by utilizing the clusters and gateways.

Consider Node i, which intends to send a data message to Node j. Node i looks up its cluster table to find the cluster ID of Node j, i.e., Ω_i^j. According to Ω_i^j, three scenarios are considered: intra-cluster routing, one-hop inter-cluster routing, and multi-hop inter-cluster routing. Note that, these three scenarios are outlined for a better understanding. In a real implementation, message forwarding from a node to one another is determined upon the meeting event between them. For instance, when Node i meets node j, i will forward messages destined to j if it is carrying any, without considering cluster information. In the mean time, i will forward Message M to j if the cluster of j is in the path of M.

A. Intra-cluster Routing

If $\Omega_i^j = \Omega(i)$, Nodes i and j are in the same cluster (see Fig. 2(a)). Since all nodes in a cluster have high contact probability, direct transmission is employed here. In other words, Node i transmits the data message only when it meets Node j. No relay node is involved in such intra-cluster routing.

B. One-hop Inter-cluster Routing

If $\Omega_i^j \neq \Omega(i)$, Node i looks up its gateway table. If an entry for Ω_i^j is found, there exists a gateway, i.e., $G_i^{\Omega_i^j}$, to Node j's cluster. In this scenario, Node i sends the data message to its gateway $G_i^{\Omega_i^j}$. Upon receiving the data message, the gateway looks up its gateway table to find Node j's cluster ID. Whenever, it meets *any* node, e.g., Node k, in Node j's cluster, it forwards the message to Node k, which in turn delivers the data message to Node j through intra-cluster routing as discussed above. Since Node $G_i^{\Omega_i^j}$ is the gateway, it has high probability to meet at least one node in Node j's cluster. Note that, Node k in Fig. 2(b) is not necessary to be the gateway node (that pairs with Node $G_i^{\Omega_i^j}$ for these two clusters).

C. Multi-hop Inter-cluster Routing

If $\Omega_i^j \neq \Omega(i)$ and Node i doesn't find Ω_i^j in its gateway table, the approaches discussed so far will fail to deliver the data message, because the destination (Node j) is not in any cluster that is reachable by Node i's gateways. As a result, the data transmission from Node i to Node j needs to be devised for multi-cluster routing.

Given the low connectivity in delay-tolerant mobile networks, on-demand routing protocols do not work effectively here, because the flooding-based on-demand route discovery leads to extremely high packet dropping probability, as shown in [6]. On the other hand, any table-driven routing algorithms may be employed for multi-hop inter-cluster routing. For simplicity, a link-state-like routing scheme is used as an example in the following discussions.

In the protocol, every gateway node builds and distributes a *Cluster Connectivity Packet (CCP)* to other gateways in the network. The CCP of Gateway i comprises its cluster ID, and a list of clusters to which it serves as the gateway and the corresponding contact probabilities, i.e., $\{(c, \zeta_i^c) \mid \forall G_i^c = i\}$. Such information can be readily obtained from the gateway table. Examples of CCP are shown in Fig. 2(c). Note that the actual implementation of CCP also includes a sequence number to eliminate outdated information.

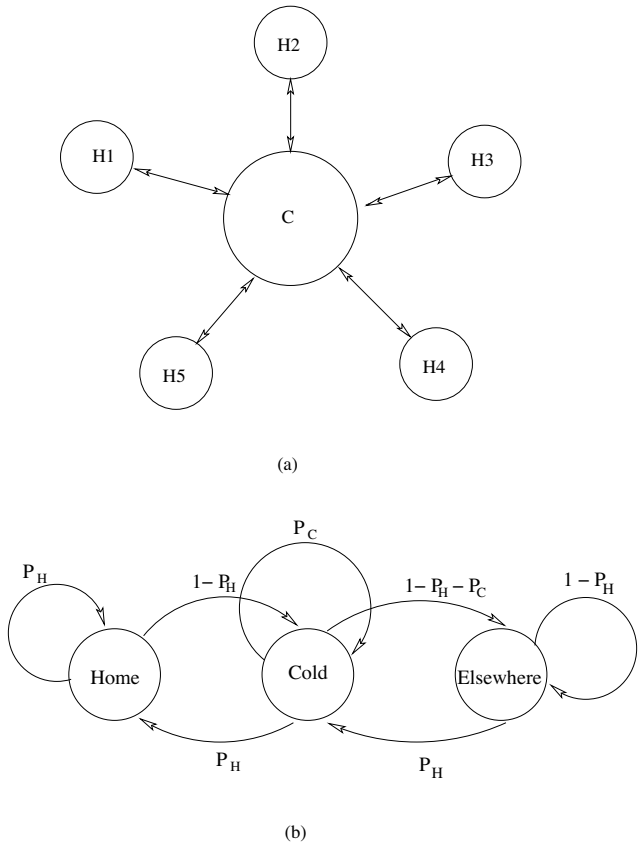


Fig. 3. Community-based mobility model.

Once a gateway node accumulates a sufficient set of CCP's, it constructs a network graph. Each vertex in the graph stands for a cluster. A link connects two vertices if there are gateways between these two clusters. The weight of the link is the contact probability of the corresponding gateway nodes. Based on the network graph, a shortest path algorithm is employed to find routing paths and establish the routing table. Each entry in the routing table consists of the ID of a destination cluster and the ID of the next hop cluster, in order to reach the destination. Examples of the routing table are also shown in Fig. 2(c).

Now let's see how to deliver the data messages according to the routing table. Again, assume Node i intends to send a data message to Node j. If Node i is a gateway (e.g. it is Node k_1 in Fig. 2(c)), it simply looks up its routing table to find the destination cluster (Ω_i^j) and the corresponding next hop cluster, denoted by C_{next}. If Node i is not a gateway, it doesn't maintain the routing table and thus has no clue about routing. As a result, it asks the first gateway node it meets for routing information, C_{next}. In either case, once C_{next} is known by Node i, one-hop inter-cluster routing is employed to send the data message to any node in C_{next}. The above procedure repeats until the data messages are delivered to the destination, Node j.

D. Load Balancing

Load balancing is an effective enhancement to the proposed routing protocol. The basic idea is to share traffic load among

cluster members in order to reduce the dropping probability due to queue overflow at some nodes. Sharing traffic inside a cluster is reasonable, because nodes in the same cluster have similar mobility pattern, and thus similar ability to deliver data messages. Whenever the queue length of a node exceeds a threshold, denoted by Λ, it starts to perform load balancing. More specifically, it randomly transmits as many messages as possible to any node it meets, until their queues are equally long or the latter's queue becomes longer than Λ.

IV. Simulations

A. Mobility Model

We adopt the community-based mobility model introduced in [13], [19], but with modifications to make it closer to reality. Herein, we suppose there are five hot spots and a cold spot denoted as $H_1 - H_5$ and C, respectively as shown in Fig. 3(a). The number of spots is chosen mainly due to the research results obtained from real user mobility traces as described in [18], [21]. We further assume that all communications can be done only by nodes in the same hot spot and there is no communication in the cold spot. The assumption is reasonable because a hot spot may be a study hall or a library where students stay together long enough for communication, while cold spot is just a transition state between two hot spots.

In the model, we also assume that each mobile node has a "home" hot spot where it spends most of its time. For example, a computer science student is likely to choose the computer science hall to be its home spot. When the node is at home, it has a probability of P_H to stay, and $1 - P_H$ to leave, in the next time slot. In our model, when a node leaves its home, it always goes to the cold spot. While at the cold spot, it might go home with probability of P_H, or stay at that state with probability of P_C ($P_C < 1 - P_H$), or move to another hot spot with probability of $1 - P_H - P_C$. Finally, when the node is at a hot spot that is not its home, it chooses to stay there with probability of $1 - P_H$ or to move back to the cold state at the probability of P_H. The transition probabilities are depicted in Fig. 3(b). In our simulations, we set $P_H = 0.7$ and $P_C = 0.1$. We now arrive at a typical Markov chain mobility model. Let $\pi_H^{(i)}$, $\pi_C^{(i)}$, and $\pi_E^{(i)}$ be the steady state probabilities of node i in home state, cold state, and elsewhere state (any other hot spot), respectively. From Markov chain theory, we have: $\pi_H^{(i)} = \frac{P_H^2}{P_H+(1-P_H-P_C)(1-P_H)}$, $\pi_C^{(i)} = \frac{P_H(1-P_H)}{P_H+(1-P_H-P_C)(1-P_H)}$, and $\pi_E^{(i)} = \frac{(1-P_H)(1-P_H-P_C)}{P_H+(1-P_H-P_C)(1-P_H)}$. Those steady probabilities, as to be discussed next, are useful indicators for the selection of clustering threshold γ as well as gateway threshold $\hat{\gamma}$.

B. Simulation results

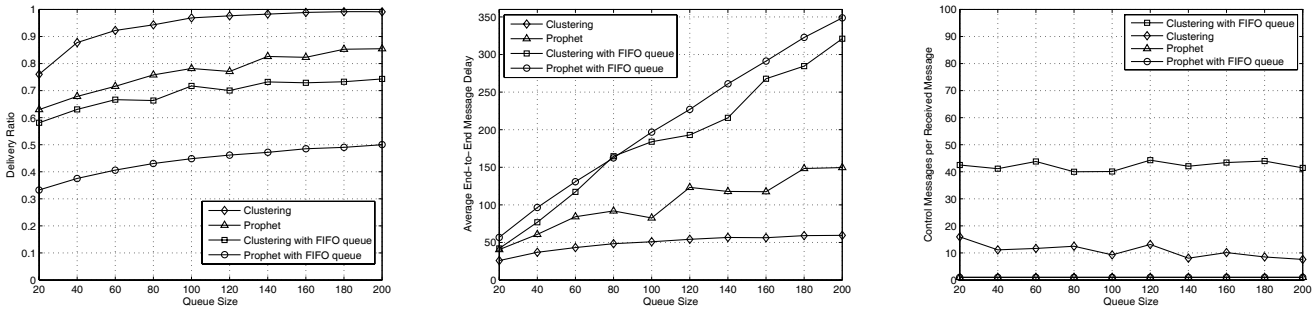
We compare our protocol with a widely known "flat" routing protocol, namely Prophet, under the same conditions given in [13] to demonstrate the effectiveness of clustering in DTN routing. We choose Prophet because it uses a similar mobility model, and is also a no-duplication data message routing protocol. Our metrics for evaluation and comparison include the message delivery ratio (i.e., the successfully delivered messages over the total generated messages), the average message end-to-end delay, and the average number of control messages for a successful delivered data message. In the simulations, we assume packet loss of 10%, due to noise/interference in wireless channel. A data message will be dropped in the following cases: (i) when it is transmitted from one node to another if it is affected by channel noise/interference, and (ii) when it reaches a new node but the queue of that node is full.

We have carried out our simulation under different scenarios to study the impact of different network parameters on the network performance. Two queue management scenarios are considered. In the first scenario, FIFO queue is employed at each node as it is originally used in Prophet. In the second scenario, we employ a simple queue management as follows. Consider Node i in meeting with Node j, (i) Node i first sends a message directly to node j if it has one destined to j regardless of cluster information; (ii) Node i sends a message to j if it has a message to another node in j's cluster, when i and j are in different clusters; (iii) Node i sends a message to j if j is a gateway to (a) adjacent cluster that contains the destination of the message or (b) j is a cluster in the path of the message (in multi-hop inter-cluster routing). Meanwhile, a similar queuing principle is adopted for Prophet: when Node i meets Node j, Node i will select a message that j has the highest probability to reach its destination to forward to j (including j itself).

We also assume data messages are generated in a Poisson process. A random destination is chosen for each message. We repeat each simulation for 5 times and show the average results. In each simulation run, the simulation time is 10000 seconds and the warm up time is 500 seconds. Note that both protocols need warm up time. Our cluster-based protocol needs the warm-up period to form clusters, while Prophet needs it to initialize delivery probabilities.

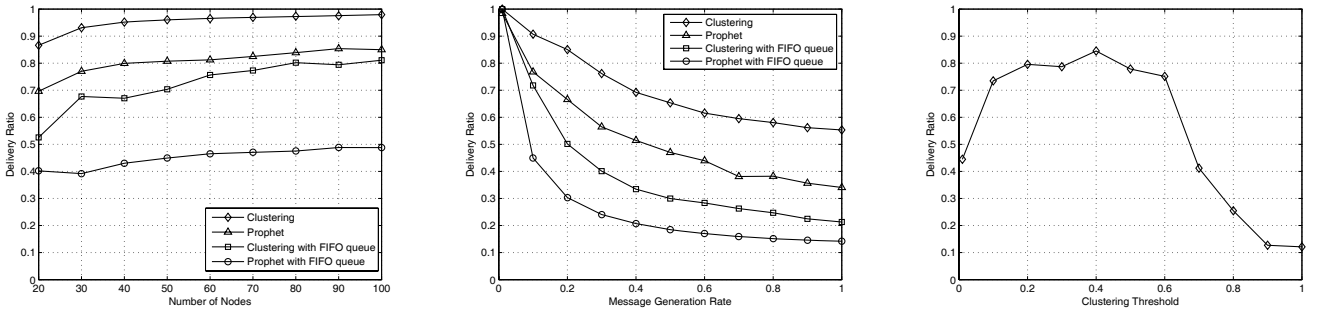
In the first simulation, we study the impact of queue size. Herein, we set the number of nodes to be 50, the data generation rate 0.1, the clustering threshold 0.6, and the gateway threshold 0.1. Every node has the same maximum queue size, which varies from 20 to 200 during the simulation. As shown in Fig. 4, our cluster-based routing protocol yields much higher overall delivery ratio (Fig. 4(a)). The higher delivery ratio is due to several reasons, including our load balancing implementation as discussed in the previous section. When the queue of a node is almost full, it will ask other members in the cluster to carry some of its data messages, resulting in low drop rate at each single node.

Fig. 4(b) shows the message end-to-end delay. As can be seen, the end-to-end delay of a message in Prophet is increasing almost linearly with the increase of queue size while cluster-based routing protocol gives shorter delay, and the increment is slower with longer queue size. The reason lies on the difference in message forwarding mechanisms. That is, in Prophet, a node always forwards its data message to the meeting node if it has higher delivery probability to the destination of that message even though its own probability to that destination is also high. The message, when arriving at the new node, needs to be queued again, leading to longer queuing time in total. In cluster-based routing, in contrary, if the node has high probability to meet the destination of the message, it will wait for a direct transmission, as described in intra-cluster routing in the previous section, to save the



(a) Delivery Ratio (b) End-to-End Delay (c) Control Message

Fig. 4. Impact of queue size.



(a) Delivery Ratio vs. Number of Nodes (b) Delivery Ratio vs. Generation Rate (c) Delivery Ratio vs. Clustering Threshold γ

Fig. 5. Impact of other parameters.

queuing time. Furthermore, for a destination in a different cluster, cluster-based routing uses cluster-to-cluster forwarding scheme, that is predictably shorter than hop-to-hop routing used in Prophet.

Another network metric for evaluation is the number of exchanged control messages in order to successfully deliver a data message. It is expected that the total control messages in the cluster-based routing protocol is higher than that of Prophet, because it needs to not only learn the contact probabilities, but also exchange clustering and gateway information. However, with higher efficiency in data delivery, the total number of delivered messages in cluster-based routing is much higher than that of Prophet. As a result, the number of control messages per successful data message of cluster-based routing is lower than that of Prophet, as shown in Fig. 4(c).

In the second simulation, we study the scalability of cluster-based routing protocol by varying the number of nodes (Fig. 5(a)) and the data message generation rate (Fig. 5(b)) while fixing the queue size to 100. Note that, in this simulation, we show only message delivery ratio instead of all three metrics since it is the most important metric and the other metrics show the same trend. Fig. 5(a) shows the average delivery ratio when the number of nodes increases from 20 to 100 (while the data generation rate is still 0.1). Both protocols tend to increase their delivery ratio when network density increases. With more nodes in network, traffic load is higher, potentially leading to bottlenecks at some nodes, thus degrading the network performance. However, if the network is balanced, the more nodes in network, the more routes to deliver messages, thus improving the network performance. In our simulation, we observe increase in delivery ratio in both protocols. However, with balancing mechanism, the delivery ratio of cluster-based routing protocol is always much higher than that of Prophet. Nevertheless, the delivery ratio in both protocols, as shown in Fig. 5(b), is degraded when the data generation rate increases. The degradations slow down when they reach a certain point, which, we believe, depends on the mobility pattern.

The last simulation is for the study of thresholds used in the protocols. γ and $\hat{\gamma}$ are two critical parameters in our cluster-based routing protocol, dictating cluster formation and gateway selection. In general, the optimization of γ and $\hat{\gamma}$ is an challenging and complicated problem, depending on several parameters such as traffic load, data queue size, and nodal contact probabilities. We plan to investigate this problem analytically in our future research. Meanwhile, we have done simulations by varying γ and $\hat{\gamma}$, in order to observe their impact on the network performance. For example, Fig. 5(c) shows the results of γ. While γ varies from 0 to 1, we observe that the maximum delivery ratio is achieved when γ is around 0.4. Now, let us revisit our discussion about community-based mobility model. We know that nodes in the same cluster usually have the same home location where they spend most of their time and learn their pair-wise contact probabilities. The average contact probability between Node i and Node j would be $p_{ij} = \pi_H^{(i)} * \pi_H^{(j)} = 0.451$ that is very close to the optimal value of γ observed in the simulation.

Although the value of γ depends on network parameters and is not universally valid, it shows the existence of global optimal. With no proof, we argue that this observation is

reasonable. More specifically, when γ is very low, all nodes are grouped into a single cluster and thus only intra-cluster transmission is enabled. As a result, the performance is likely degraded, since some nodes may have very low contact probability and thus are unable to transmit data to each other without relay. If γ is high, almost every node claims itself as a cluster. Consequently, it turns into the non-clustering approach, which has lower performance as we discussed above. Similar observation is also obtained for $\hat{\gamma}$. Thus the results are omitted.

V. Conclusion

We have investigated clustering and cluster-based routing in DTMN. The basic idea is to let each mobile node to learn unknown and possibly random mobility parameters and join together with other mobile nodes that have similar mobility pattern into a cluster. The nodes in a cluster can then interchangeably share their resources for overhead reduction and load balancing in order to improve overall network performance. Due to the lack of continuous communications among mobile nodes and possible errors in the estimated nodal contact probability, convergence and stability become major challenges in distributed clustering in DTMN. To this end, an exponentially weighted moving average (EWMA) scheme has been employed for on-line updating the contact probabilities, with its mean proven to converge to the true contact probability. Based on contact probabilities, a set of functions including *Sync()*, *Leave()*, and *Join()* has been devised for cluster formation and gateway selection. Finally, the gateway nodes exchange network information and perform routing. Extensive simulations have been carried out to evaluate the efficiency of the proposed cluster-based routing protocol. The results have shown that it achieves higher delivery ratio and significantly lower overhead and end-to-end delay, compared with its non-clustering counterpart.

References

[1] K. Fall, "A delay-tolerant network architecture for challenged Internets," in *Proc. ACM SIGCOMM*, pp. 27–34, 2003.

[2] S. Burleigh, A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K. Scott, and H. Weiss, "Delay-tolerant networking—an approach to interplanetary Internet," *IEEE Commun. Mag.*, vol. 41, no. 6, pp. 128–136, 2003.

[3] http://www.princeton.edu/ mrm/zebranet.html.

[4] T. Small and Z. J. Haas, "The shared wireless infostation model: a new ad hoc networking paradigm (or where there is a whale, there is a way)," in *Proc. MobiHOC*, pp. 233–244, 2003.

[5] T. Small and Z. J. Haas, "Resource and performance tradeoffs in delay-tolerant wireless networks," in *Proc. ACM SIGCOMM Workshop on Delay Tolerant Networking and Related Topics*, pp. 260–267, 2005.

[6] Y. Wang and H. Wu, "DFT-MSN: the delay fault tolerant mobile sensor network for pervasive information gathering," in *Proc. 26th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'07)*, pp. 1235–1243, 2006.

[7] Y. Wang and H. Wu, "Delay/fault-tolerant mobile sensor network (DFT-MSN): a new paradigm for pervasive information gathering," *IEEE Trans. Mobile Computing*, vol. 6, no. 9, pp. 1021–1034, 2007.

[8] Y. Wang, H. Wu, F. Lin, and N.-F. Tzeng, "Cross-layer protocol design and optimization for delay/fault-tolerant mobile sensor networks," *IEEE J. Sel. Areas Commun*, vol. 26, no. 5, pp. 809–819, 2008. (A preliminary version was presented at IEEE ICDCS'07.)

[9] H. Wu, Y. Wang, H. Dang, and F. Lin, "Analytic, simulation, and empirical evaluation of delay/fault-tolerant mobile sensor networks," *IEEE Trans. Wireless Commun.*, vol. 6, no. 9, pp. 3287–3296, 2007.

[10] M. Musolesi, S. Hailes, and C. Mascolo, "Adaptive routing for intermittently connected mobile ad hoc networks," in *Proc. IEEE 6th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WOWMOM)*, pp. 1–7, 2005.

[11] J. LeBrun, C.-N. Chuah, and D. Ghosal, "Knowledge based opportunistic forwarding in vehicular wireless ad hoc networks," in *Proc. IEEE Vehicular Technology Conference (VTC) Spring*, pp. 1–5, 2005.

[12] W. Zhao, M. Ammar, and E. Zegura, "A message ferrying approach for data delivery in sparse mobile ad hoc networks," in *Proc. MobiHOC*, pp. 187–198, 2004.

[13] A. Lindgren, A. Doria, and O. Scheln, "Probabilistic routing in intermittently connected networks," in *Proc. First International Workshop on Service Assurance with Partial and Intermittent Resources*, pp. 239–254, 2004.

[14] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot, "Pocket switched networks and human mobility in conference environments," in *Proc. ACM SIGCOMM Workshop on DTN and Related Topics*, pp. 244–251, 2005.

[15] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Spray and wait: an efficient routing scheme for intermittently connected mobile networks," in *Proc. ACM SIGCOMM Workshop on DTN and Related Topics*, pp. 252–259, 2005.

[16] C. Liu and J. Wu, "Scalable routing in delay tolerant networks," in *Proc. ACM MobiHoc*, 2007.

[17] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott, "Impact of human mobility on the design of opportunistic forwarding algorithms," in *Proc. IEEE INFOCOM*, pp. 1–13, 2006.

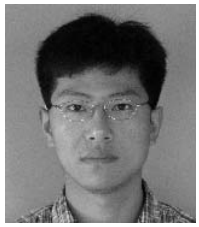
[18] M. Kim, D. Kotz, and S. Kim, "Extracting a mobility model from real user traces," in *Proc. IEEE INFOCOM*, pp. 1–13, 2006.

[19] T. Spyropoulos, K. Psounis, and C. Raghavendra, "Performance analysis of mobility-assisted routing," in *Proc. MobiHoc '06: 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pp. 49–60, 2006.

[20] J. Leguay, T. Friedman, and V. Conan, "DTN routing in a mobility pattern space," in *Proc. WDTN '05: 2005 ACM SIGCOMM Workshop on Delay-tolerant Networking*, pp. 276–283, 2005.

[21] http://www.cs.dartmouth.edu/ campus/data.html.

Ha Dang received his B.E. degree in Computer Engineering from Hanoi University of Technology in 1999 and M.S. degree from University of Louisiana at Lafayette in 2005. He earned his Ph.D. in Computer Science from The Center for Advanced Computer Studies, University of Louisiana at Lafayette in 2009. His research interests include mobile ad hoc networks, wireless sensor networks, and distributed systems.



Hongyi Wu (M'02) received his Ph.D. degree in Computer Science and M.S. degree in Electrical Engineering from State University of New York (SUNY) at Buffalo in 2002 and 2000, respectively. He received his B.S. degree in Scientific Instruments from Zhejiang University in 1996. He is currently an Associate Professor at The Center for Advanced Computer Studies (CACS), University of Louisiana (UL) at Lafayette. His research interests include wireless mobile ad hoc networks, wireless sensor networks, next generation cellular systems, and integrated heterogeneous wireless systems. He received NSF CAREER Award in 2004.