Hindawi
Wireless Communications and Mobile Computing
Volume 2018, Article ID 3806907, 19 pages
https://doi.org/10.1155/2018/3806907

Research Article

Learning Automata Based Caching for Efficient Data Access in Delay Tolerant Networks

Zhenjie Ma, Haoran Wang, Ke Shi, and Xinda Wang

College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

Correspondence should be addressed to Ke Shi; keshi@mail.hust.edu.cn

Received 26 September 2017; Revised 11 January 2018; Accepted 29 January 2018; Published 5 March 2018

Academic Editor: Daniele Pinchera

Effective data access is one of the major challenges in Delay Tolerant Networks (DTNs) that are characterized by intermittent network connectivity and unpredictable node mobility. Currently, different data caching schemes have been proposed to improve the performance of data access in DTNs. However, most existing data caching schemes perform poorly due to the lack of global network state information and the changing network topology in DTNs. In this paper, we propose a novel data caching scheme based on cooperative caching in DTNs, aiming at improving the successful rate of data access and reducing the data access delay. In the proposed scheme, learning automata are utilized to select a set of caching nodes as Caching Node Set (CNS) in DTNs. Unlike the existing caching schemes failing to address the challenging characteristics of DTNs, our scheme is designed to automatically self-adjust to the changing network topology through the well-designed voting and updating processes. The proposed scheme improves the overall performance of data access in DTNs compared with the former caching schemes. The simulations verify the feasibility of our scheme and the improvements in performance.

1. Introduction

Delay Tolerant Networks (DTNs) [1] are proposed as a network architecture to address communication issues in challenging environments where network connectivity is subject to frequent and lasting disruptions. DTNs consist of a number of communicating devices which contact each other opportunistically so that only intermittent connectivity exists in DTNs. As a result, DTNs are normally characterized by unpredictable node mobility, high forwarding latency, and the lack of global network state information. To address the data access in DTNs, "carry-and-forward" mechanism is used for data transmission in DTNs. When transmitting data, each mobile node acts as a relay to store the passing data and forward the data when contacting other nodes.

DTNs have been introduced into many recent applications. For instance, users with personal mobile devices utilize mobile P2P networks to share data in a certain area [2]. DTNs can also be used in mobile military communication networks to deliver real-time battlefield information locally [3]. In these applications, mobile nodes not only request data but also generate data themselves. Meanwhile, all mobile nodes are also responsible for forwarding passing data. It is necessary to design an appropriate network scheme to coordinate all mobile nodes to cache and forward data in DTNs.

Different caching techniques have been widely used to improve data access performance in many studies. The basic idea of caching techniques in networks is to store data at appropriate locations such that future requests for the data can be replied promptly. Although caching techniques have been extensively studied in traditional networks and wireless networks, they are rarely applied to DTNs considering the harsh network environments. The intermittent connectivity leads to the difficulty of determining the appropriate set of caching nodes in networks. And it is also hard to determine and maintain a set of caching nodes to adapt to the changing network topology given the unpredictable node mobility.

To address the data caching and access problems in DTNs, a number of caching schemes have been proposed. Some of the schemes [4] select a set of caching nodes based on probabilistic metrics, and others [5] use Markov chain to predict the nodes mobility and reduce data access delay.

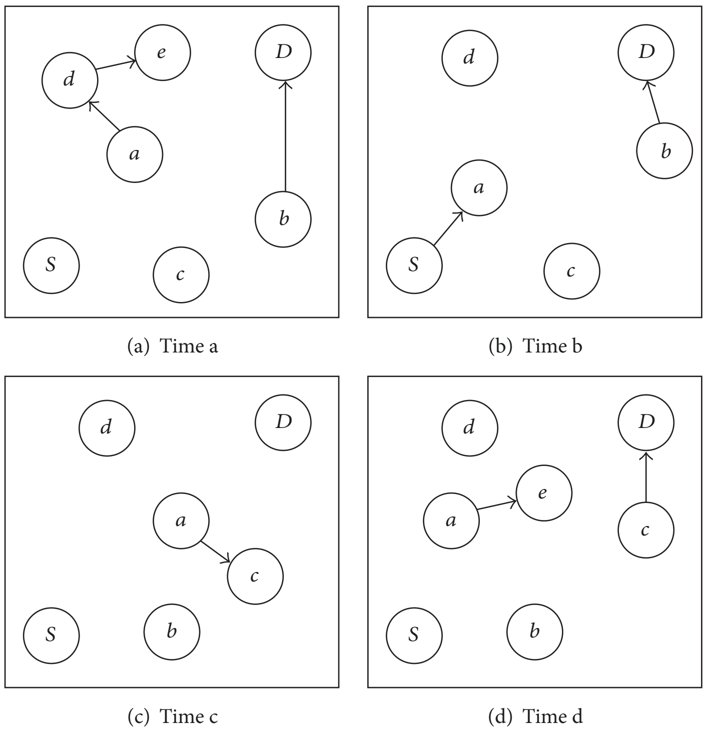


Figure 1: DTNs networks model and data transmission.

However, most existing schemes are still less practicable since the required global network state information is hard to obtain and may consistently change in DTNs.

In order to overcome the aforementioned issues in DTNs, we propose a novel data caching scheme based on the distributed learning automata. Our basic idea is to select and maintain a set of caching nodes called Caching Node Set (CNS), which caches data and addresses data requests from other nodes. When nodes generate new data, new data will be disseminated to all caching nodes intentionally. Besides, CNS can be updated in real time to adapt to the network topology changes via learning automata mechanism. With the updating of CNS, data will be redistributed among the latest caching nodes by cache replacement strategies. The major contributions in the paper are listed as follows:

(i) We propose a novel caching scheme to coordinate multiple caching nodes for addressing data access and to improve the overall performance of networks in DTNs.

(ii) We propose a novel algorithm utilizing distributed learning automata to construct the optimal Caching Node Set (CNS) to address the network topology change in DTNs. To maintain CNS in real time, two well-designed processes, voting process and updating process, are introduced to our scheme.

(iii) We propose a distributed algorithm requiring no global network state information to fulfill our scheme, which improves the practicability of our scheme for DTNs-bases applications.

The remainder of the paper is organized as follows. Section 2 describes the basic idea of our approach; Section 3 describes design and implementation of learning automata based caching node selection; Section 4 describes cache based data access; Section 5 presents our evaluation; Section 6 reviews related work; Section 7 concludes the article; and Section 8 gives a glimpse of further works in this area.

2. Setup

2.1. Problem Statement. DTNs can be described by network contact graph $G(V(t), E(t))$, where each vertex in $V(t)$ represents a network node in DTNs at time t and each edge e_{ij} represents a contact between node i and node j at time t. Due to the node mobility, some nodes may leave the network while new nodes may join; thus $V(t)$ changes along with time. Similarly $E(t)$ represents opportunistic contacts in DTNs and changes along with time as well, so edge e_{ij} only exists when the node pair $i, j \in V(t)$ contacts each other at time t; otherwise edge e_{ij} does not exist.

We assume that when node i and node j contact each other at time t, that is, edge e_{ij} exists at time t, data can be forwarded from node i to node j. The network model and an example of data transmission in DTNs are shown in Figure 1. The data are assumed to be transmitted from node S to node D.

In Figure 1(a), at time t_1, there are six nodes in the network, a, b, c, d, e, S, and D. And there are three edges, e_{ad}, e_{de}, and e_{bD}, which indicates that only these three node pairs can transmit data in the network.

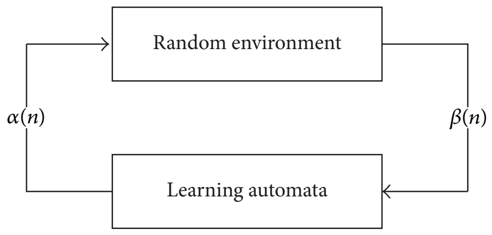


FIGURE 2: Relationship between learning automata and random environment.

In Figure 1(b), at time t_2, node S generates new data and prepares to transmit it to node D. Since there is no direct end-to-end path from node S to node D at time t_2, node S has to forward the data to node a that relays the data to node D. Node a has to cache the data because it cannot contact other nodes and forward the data immediately. Meanwhile the network topology changes and node e leaves the network.

In Figure 1(c), after a while, at time t_3, node a moves and contacts node c. Then node a forwards the data to node c and node c will cache the data until it contacts other nodes.

In Figure 1(d), at time t_4, node c contacts node D, then node c forwards the data to node D, and the data transmission finishes. Meanwhile node c may move and join the network again.

Therefore, to improve the performance of data access, we need to find a shortest path between the requesting node and the hosting node. Here the shortest path means the path with the highest data delivery probability, which is implemented through the contacts between the nodes along this path.

2.2. Learning Automata. A learning automaton [6] is a self-adaptive unit designed to achieve certain goals by learning through repeating interactions with outer random environments following a predefined sequence of rules. In the learning process, the learning automaton chooses the optimal actions from a finite set of actions based on a probability distribution in each instant. For each action taken by the learning automaton, the environment will respond with a reinforcement signal. Then the learning automaton updates its action probability vector according to the feedback signal. The relationship between the learning automaton and outer random environment is shown in Figure 2. The objective of learning automata is to evolve progressively to a desired state.

Learning automata can be classified into two main structures: fixed structure learning automata and variable structure learning automata. Learning automata with variable structure are represented by a triple $\langle \alpha, \beta, T \rangle$ where α denotes the set of actions, β denotes the set of feedback signals, and T denotes the learning algorithm. The learning algorithm is a recurrence relation used to modify the state probability vector. Let $\alpha_i(k)$ and $p(k)$ denote the action chosen at time k and the action probability vector on which the chosen action is, respectively, based. The recurrence equation for updating the action probability vector p is shown by (1) and (2).

$$p_j(k+1) = \begin{cases} p_j(k) + a \cdot \left(1 - p_j(k)\right) & j = i \\ (1-a) \cdot p_j(k) & \forall j j \neq i \end{cases} \quad (1)$$

when the taken action is rewarded by the environment (i.e., $\beta(n) = 0$) and

$$p_j(k+1) = \begin{cases} (1-b) \cdot p_j(k) & j = i \\ \dfrac{b}{r-1} + (1-b) \cdot p_j(k) & \forall j j \neq i \end{cases} \quad (2)$$

when the taken action is penalized by the environment (i.e., $\beta(n) = 1$). r is the number of actions from which the learning automaton can choose.

a and b in the recurrence equations denote the reward and penalty parameters. If $a = b$, the learning algorithm is called linear reward-penalty ($L_{R\text{-}P}$) algorithm. If $a \gg b$, it is called linear reward-ε ($L_{R\text{-}\varepsilon P}$) penalty algorithm. And if $b = 0$, it is called linear reward-inaction ($L_{R\text{-}I}$) algorithm [7].

Distributed learning automata consist of a number of learning automata which form a network and achieve a global optimal result cooperatively. Each learning automaton in the network updates its own action probability vector based on the feedback signals received from neighboring learning automata.

2.3. Main Idea. Our basic idea is to let the nodes decide whether to cache data automatically using learning automata mechanism. The nodes that cache the data are selected as caching nodes and construct the Caching Node Set (CNS) cooperatively. In our proposed scheme, each network node is assigned a learning automaton. When DTNs start operating, all learning automata in nodes are activated and start constructing the CNS to cache data and address data requests based on the node state information. Following rules in the learning automata algorithms, a number of easily accessed nodes in DTNs will become the caching nodes and comprise the CNS together. And more importantly, to address the changing network environments in DTNs, CNS can update itself to adapt to the network topology changes in real time utilizing the learning automata.

The construction and maintenance of CNS are shown in Figure 3. For each node, the action set of learning automata includes two actions, setting as a CNS node and setting as a non-CNS node. At the beginning, the node chooses not to be a CNS node. After that, a set of nodes making frequent contacts with other nodes is selected as CNS, which corresponds to the gray nodes in Figure 3(a). All other nodes in DTNs are able to contact a node in CNS at least once in a period of time with high probability. Node b and node g are selected as caching nodes, and arrow lines indicate the opportunistic contacts among nodes. It is necessary to ensure that nodes a, d, e, c, h, and f can contact node b or node g in a certain period of time. When networks topology changes along with time, the CNS is updated to cover all nodes in network as much as possible. In Figure 3(b), with the mobility

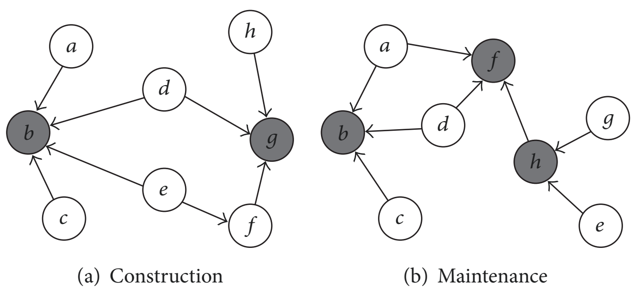


Figure 3: The construction and maintenance of CNS.

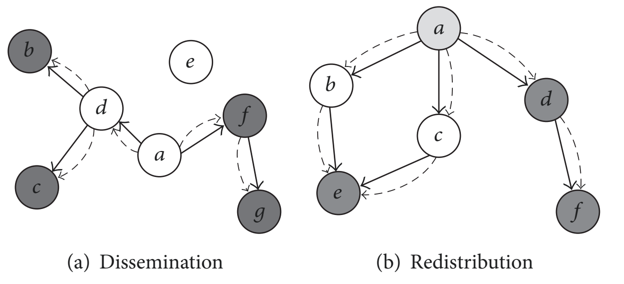


Figure 4: The dissemination and redistribution of data.

of nodes and variability of contacts among nodes, node f and node h are selected as new caching nodes while node g ceases to be a caching node.

When CNS is constructed, we focus on utilizing the available node buffer to improve the performance of data access. When a node generates new data with the globally unique identifier and a finite lifetime, the data copies will be disseminated to all caching nodes as quickly as possible. When a node requests for data, it will send the queries to neighboring caching nodes to pull data. If the caching nodes have the data copies other nodes request, the request are replied; otherwise the request fails.

The dissemination of data is shown in Figure 4(a), where the dash lines indicate the transmission of data. New data are generated by node a and then disseminated to all caching nodes using the data forwarding strategy. When CNS changes or the nodes deplete the storages, cached data will be replaced and redistributed to ensure data accessibility. The data replacement is shown in Figure 4(b). When node a ceases to be a caching node, data cached in node a are removed and redistributed to other caching nodes. The overall performance of a caching scheme relies on prompt data dissemination and optimal data replacement strategies.

3. Learning Automata Based Caching Node Selection

In this section, we discuss how to construct and maintain CNS based on distributed learning automata.

3.1. Preview. To introduce our new scheme, we first make a preview and start with the presentation of several new concepts and terms used in our scheme.

3.1.1. Votes. The vote is a kind of message sent from one node to another in our scheme. The votes used in our scheme are assigned different weight values, which are used to select the caching nodes in networks. There are totally *VOTE_K* different possible weight values which can be assigned to all votes. According to expected utility principle, the weight values of votes can also be regarded as the expected quantity of transmitted data. The weight values of votes are decided by the frequency of node contacts and *state*, referring to the sum of the votes weight value a node receives.

The *VOTE_K* different possible weight values indicate that there are *VOTE_K* different possible choices of nodes to which a node can transmit the data ordered by the vote weight values from high to low. When transmitting data, the node prefers to transmit the data to the node u_1 which receives the greatest vote weight value. If node u_1 is unreachable or disabled, the node would transmit the data to the node u_2 with the second greatest vote weight value and so on. In this way, we can improve the successful rate of data access, decrease the loss of data, and reduce the transmission delay.

Besides direct contacts between two nodes, data in DTNs may be transmitted from the starting node to the destination node through multiple contacts. Therefore, we present two different types of votes, direct votes and indirect votes. When node A votes for node B, the direct vote is the vote that node B receives from node A and the weight value of the direct vote *direct_vote* is added to *state* of node B directly. And the indirect vote is the vote node A relays to another neighboring node C through node B. Generally, node C is unaware of the vote from node A until node B sends it to node C. Therefore node B has to cache the indirect vote until node B contacts node C. When receiving the indirect vote, node C adds the weight value of the indirect vote *indirect_vote* to *state*.

Voting plays an essential part in caching nodes selection. However, only direct voting between two nodes in contact is inadequate; we propose indirect voting to refine the scheme. The main purpose of indirect voting is to intentionally redirect the votes weight values to the nodes that tend to be caching nodes, which increases the difference of *state* values of all nodes and makes it more conducive to selecting an appropriate Caching Node Set. Otherwise, the votes may be dispersed among all the nodes uniformly. The uniform distribution of *state* values may increase the difficulty of caching nodes selection and the instability of the Caching Node Set, causing the loss of network performance.

3.1.2. Node Information. Beside the passing data caching nodes need to store to improve data transmission; to implement our scheme, each node also needs to cache necessary and relevant node information to build and maintain our scheme. The following node information is cached in each node:

(i) *state* indicates the sum of the votes weight value a node has received.

(ii) *dor_prob* indicates the action probability of setting a node as a caching node. Correspondingly, 1 − *dor_prob* indicates the probability of setting a node as a noncaching node. Whether a node is a caching node or not is finally determined by the value of *dor_prob*.

(iii) *M_contacts_rec* records the frequency of contacts with other nodes in history.

(iv) *M_adj_node* records node information of other nodes.

(v) *M_indirect_vote* records the indirect votes' values for other nodes.

(vi) *Q_max_k_heap* sorts the neighboring nodes ordered by the frequency of contacts.

(vii) *adj_max_state* records the greatest *state* value of all neighboring nodes.

(viii) *adj_max_node* records the node ID with the greatest *state* value of all neighboring nodes.

(ix) *contacts* records the frequency of contacts with the node *adj_max_node*.

(x) *is_dominator* serves as an indicator of whether the node is a caching node.

More specifically, a node allocates a queue to store all necessary information of each contacted node, including the node information *M_adj_node*, the frequency of contacts *M_contacts_rec*, and the indirect vote information to the node *M_indirect_vote*. Ideally, all the relevant information can be stored in the queue permanently; however, a network node only provides limited memory for information storage. Therefore, the restricted memory allocation for the queue leads to eliminating obsolete information stored in queue. We first assign a time stamp to each queue item; the time stamp of the certain queue item storing the contacted node information is updated every time the node make a contact. The queue always eliminates the items with the outdated time stamps when the memory has depleted. On the other hand, each queue item has a predefined lifetime and will be removed when it expires.

3.1.3. Information Delivery. A node will transmit the node information when contacting other nodes besides voting. When node A contacts node B, node A delivers the node information of its own to node B. When receiving the information, node B would record and update the cached information of node A. When delivering the node information, node A increases the *M_contacts_rec*[B] by 1. *MSG_REC* is the message containing the node information and contains the following details:

(i) *state* indicates the total weight value of votes a node receives.

(ii) *adj_max_state* records the greatest *state* value of all neighboring nodes.

(iii) *adj_max_node* records the node ID with the greatest *state* value of all neighboring nodes.

(iv) *contacts* records the frequency of contacts with the node *adj_max_node*.

(v) *is_dominator* indicates whether the node is a caching node.

When receiving the message from node A, node B records the information of node A in *M_adj_node*. Moreover, node B updates its *state* value according to the received information. The direct votes' value from node A is added to *state* and the indirect votes from node A to other nodes are cached in *M_indirect_vote*. Node B updates *adj_max_state* and *adj_max_node* as well according to the *state* value of node B. $B.adj\_max\_state = max(B.adj\_max\_state, B.state, A.state)$, and $B.adj\_max\_node$ is updated as the node ID which has the greatest *state* value.

3.2. Overview. The proposed CNS selecting process based on learning automata starts simultaneously with the network operation and is aimed at selecting a set of nodes as caching nodes to construct CNS and updating CNS in real time to adapt to the change of network topology. It consists of two processes: voting process and updating process, as shown in Figure 5.

The action set of the node includes two actions, setting as a CNS node and setting as a non-CNS node. The node selects its action based on *dor_prob* value. The initial *dor_prob* is set to 0.5 for each node. Once the action is selected, some nodes become the CNS node. It affects the following voting process. The votes that the nodes get change and the state values also change. This leads to new reinforcement signal and new rewarding/penalizing parameter and then new action probability *dor_prob*.

In the voting process, when node A makes a contact with node B, node A starts a voting process which sends the vote of different weight values to node B according to the node information and connection conditions between each other. When receiving the vote from node A, node B updates its node information. Meanwhile, node A sends a message containing its own node information to node B in the voting process as well. And when receiving the message from node A, node B updates the cached node information of node A. When delivering message, node A finishes the voting process. Node B finishes the voting process when receiving the message and then updates its own node information.

In the updating process, every node in the network updates its own node state information and *state* value according to the cached node information of neighboring nodes. Each node repeats the updating process at a predefined time interval independently. Equipped with a learning automaton, each node updates the action probability vector based on the reward-penalty algorithm and reinforcement feedback from other nodes. Each node will decide whether to be a caching node based on the action probability vector before finishing the updating process.

These two kinds of processes occur independently. The voting process occurs whenever the node contacts other nodes, and the updating process repeats at a predefined time interval independently. One kind of process is affected by the feedback information from the other. The updating process updates the node state information according to the votes and information messages received in the voting process, while the voting process decides the vote values according to the node state changes in the updating process. The time line of one single node is shown in Figure 6 where v_i indicates that

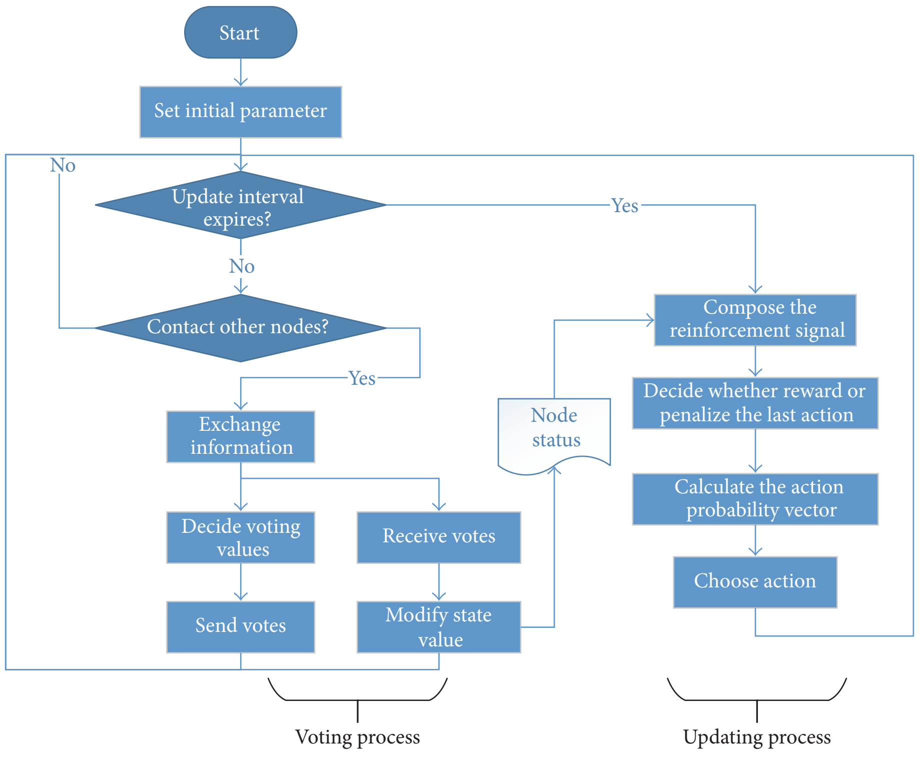


FIGURE 5: The procedure of node using our scheme.

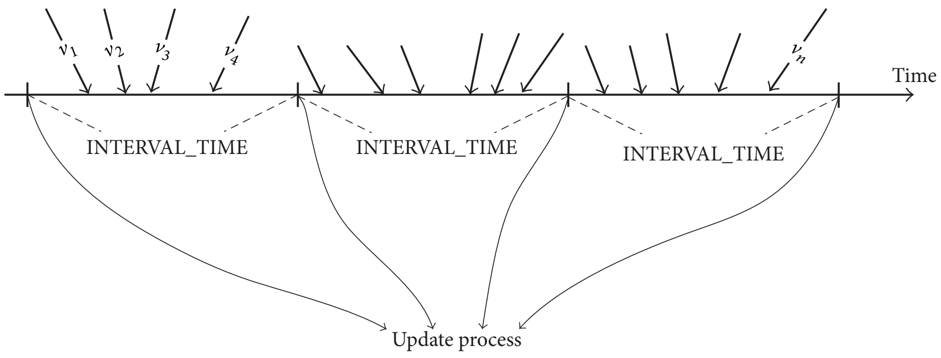


FIGURE 6: The timeline of node using our scheme.

the node makes a contact with another neighboring node and that a voting process occurs. *INTERVAL_TIME* represents the time interval at which the updating processes occur.

3.3. Voting Process. The voting process occurs whenever a contact happens. When node A contacts node B, the pair of nodes starts a voting process. A voting process consists of two parts, sending votes and delivering node information. However, the voting processes which happened in node A and node B are different. Node A, which starts a contact, votes for node B and delivers the node information to node B, while node B receives the votes from node A and updates the stored node information of node A.

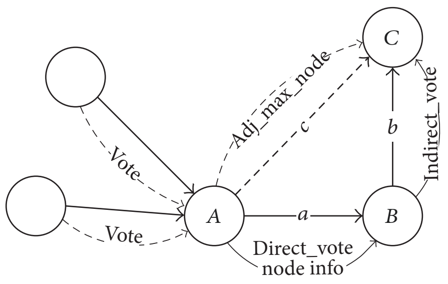


FIGURE 7: Voting process.

A voting process is shown in Figure 7 where node A contacts and votes for node B. Generally, three node, node A, node B, and node C, are involved in a voting process. And we assume $B.adj\_max\_node = C$ and $B.adj\_max\_state =$

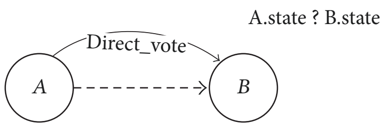


Figure 8: The relationship between *A.state* and *B.state* is uncertain.

C.state. The solid lines indicate contacts between two nodes in history, while the dashed line indicates possible contacts between each other. The values including a, b, and c on the lines indicate the accumulative number of contacts in history. And *direct_vote* indicates the direct vote and *indirect_vote* indicates the indirect vote; the directions of arrows indicate which nodes vote and which ones receive votes. Generally speaking, the nodes prefer to give their votes to the nodes that have a high possibility (depicted as state) to be CNS node and they contact a lot.

According to the difference of the node information, there are mainly three different scenarios where the distribution of vote weight values is different.

Scenario 1 (the relationship between *A.state* and *B.state* is uncertain). In the first scenario, node A fails to realize the presence of node B; that is, *A.M_adj_node* contains no information of node B. It is mainly because that node B has never made a contact with node A and node A receives no message and data from node B. In other words, node A is unaware of *B.state* and *B.adj_max_node*. This scenario is shown in Figure 8.

This kind of scenario usually happens in the initial time of networks, when there are only a few contacts between nodes and nodes lack information of neighboring nodes. Along with the operation of networks and the dissemination of nodes information, the occurrence of the first scenario will decrease.

In this scenario, when node A contacts node B, node A increases the *M_contacts_rec*[B] by 1, which records the accumulative number of contacts from node A to node B in history. Then we determine the total votes' weight value. We assume the place of node B in *A.Q_max_k_heap* is *pos*. Finally the total votes weight value k of vote from node A to node B is *Q_vote*[*pos*] if $pos < VOTE\_K$; otherwise $k = 0$.

$$\begin{aligned} direct\_vote &= k \\ indirect\_vote &= 0. \end{aligned} \tag{3}$$

In the first scenario, the vote sent from node A to node B is the direct vote alone and the value of the indirect vote is 0. Node A sends the vote and the node information to node B, while when $k = 0$ node A only sends the node information. The equations for the distribution of votes' values are shown by (3).

Scenario 2 (*A.state* $\leq$ *B.state*). When node A is aware of the node information of node B, which is contained in *A.M_adj_node*, the relationship between *A.state* and *B.state* is clear and node A is also aware of the node information of node C, which is represented by *B.adj_max_node*, referring to the node that has the greatest *state* value in the neighborhood.

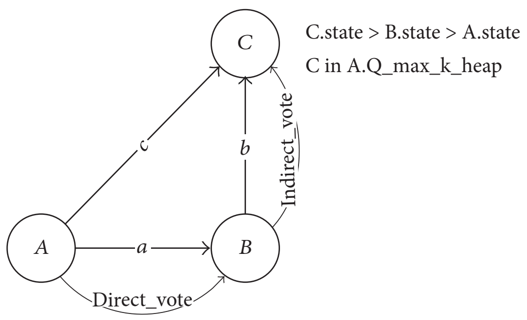


Figure 9: *C.state* > *B.state* > *A.state* and $C \in$ *A.Q_max_k_heap*.

In the second scenario, we assume *A.state* $\leq$ *B.state*. Similar to the first scenario, the total weight value of votes k is determined at first. Then the distribution of the direct vote weight value and the indirect vote weight value can be decided according to the accumulative number of contacts between nodes.

Node C has the greatest *state* value of all neighboring nodes; therefore the relationship of *state* values among node A, node B, and node C is shown

$$C.state \geq B.state \geq A.state. \tag{4}$$

According to (4), the second scenario can be classified into a number of following subscenarios.

Subscenario 2.1 (*C.state* > *B.state*). If node C is in *A.Q_max_k_heap*, it indicates that node A has contacted node C before and the number of contacts is relatively large. According to the contacts history, we can predict that node A tends to contact node C in the future. Thus, node A is more inclined to have the data cached in node C and the data transmitted from node A to node B also tends to be forwarded to node C and cached. So the contact between node A and node B can be also considered as a contact from node A to node C and at the same time, *A.M_adj_rec*[C], which records the number of contacts from node A to node C, is also increased by 1. This situation is shown in Figure 9. Then the distribution of the direct vote value and the indirect vote value can be determined.

The value of the direct vote sent from node A to node B is shown by

$$direct\_vote = \begin{cases} k \cdot \dfrac{a}{a+c} \cdot \dfrac{a-b}{a} = k \cdot \dfrac{a-b}{a+c} & a > b \\ 0 & a \leq b. \end{cases} \tag{5}$$

The value of the indirect vote relayed from node A to node C through node B is shown by

$$indirect\_vote = \begin{cases} k \cdot \left(\dfrac{c}{a+c} + \dfrac{b}{a} \cdot \dfrac{a}{a+c} \right) = k \cdot \dfrac{b+c}{a+c} & a > b \\ k & a \leq b. \end{cases} \tag{6}$$

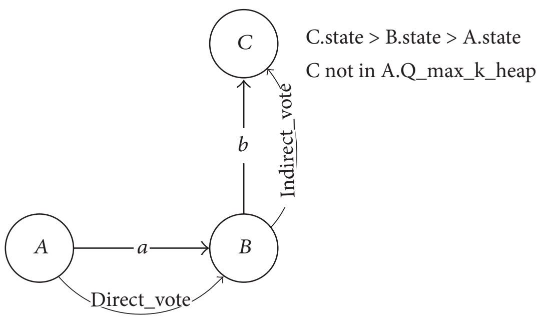


Figure 10: $C.state > B.state > A.state$ and $C \notin A.Q\_max\_k\_heap$.

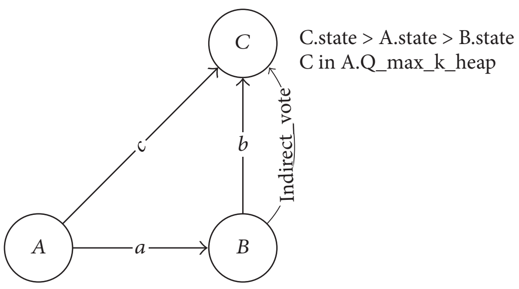


Figure 12: $C.state > A.state > B.state$ and $C \in A.Q\_max\_k\_heap$.

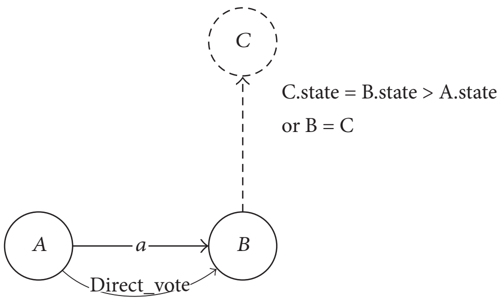


Figure 11: $C.state = B.state > A.state$ or $B = C$.

It can be argued that the *state* value decides the directions in which data are transmitted and the votes' value indicates the expectations of the quantity of transmitted data.

If node C is not in $A.Q\_max\_k\_heap$, it indicates that node A hardly or never contacts node C. Thus $A.M\_adj\_rec[C]$ remains unchanged and meanwhile the distribution of the direct vote value and the indirect vote value can be determined. This situation is shown in Figure 10.

The value of the direct vote sent from node A to node B is shown by

$$direct\_vote = \begin{cases} k \cdot \dfrac{a-b}{a} & a > b \\ 0 & a \le b. \end{cases} \quad (7)$$

The value of the indirect vote relayed from node A to node C through node B is shown by

$$indirect\_vote = \begin{cases} k \cdot \dfrac{b}{a} & a > b \\ k & a \le b. \end{cases} \quad (8)$$

Subscenario 2.2 ($C.state = B.state$). This subscenario is shown in Figure 11. According to the relationship between node C and node B, it can be classified into two situations:

(i) $B.adj\_max\_node = B$. The *state* value of node B is the greatest in the neighborhood; that is, $B.state = B.adj\_max\_state$.

(ii) $B.adj\_max\_node \neq B$. The *state* value of node B is the same as that of node C, which means $C.state = B.state = B.adj\_max\_state$.

Since $B.state$ is always equivalent to $C.state$ in these two situations; node A would tend to have the data cached in node B rather than in node C in this contact. The distributions of vote values in these two situations are the same. When the total vote value k is decided, the distribution of the direct vote value and the indirect vote value can be determined. The equations for the distribution of votes' values are shown by

$$\begin{aligned} direct\_vote &= k \\ indirect\_vote &= 0. \end{aligned} \quad (9)$$

Scenario 3 ($A.state > B.state$). Like the second scenario, in the third scenario, node A has contacted node B and is aware of the node information of node B. According to the relationship between $A.state$ and $C.state$, it can be classified into following situations:

(i) $C.state > A.state$. Node A and node C are two different nodes.

(ii) $C.state = A.state$. Node A and node C refer to the same node or are different nodes but of the same *state* value.

(iii) $C.state < A.state$: $B.adj\_max\_node$ will be updated as node A; that is, node A and node C refer to the same node.

The relationship of *state* values of node A, node B, and node C is shown by

$$C.state \ge A.state > B.state. \quad (10)$$

According to (10), the third scenario can be classified into a number of following subscenarios.

Subscenario 3.1 ($C.state > A.state$). According to the relationship between node A and node C, it can be classified into two situations.

If node C is in $A.Q\_max\_k\_heap$, this situation is shown in Figure 12. Similarly, $A.M\_adj\_rec[C]$, which records the number of contacts from node A to node C in node A, is increased by 1. When the total value of votes k is decided,

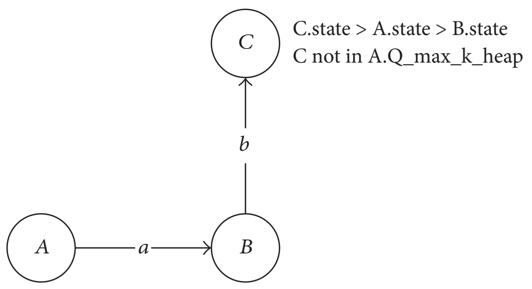


Figure 13: $C.state > A.state > B.state$ and $C \notin A.Q\_max\_k\_heap$.

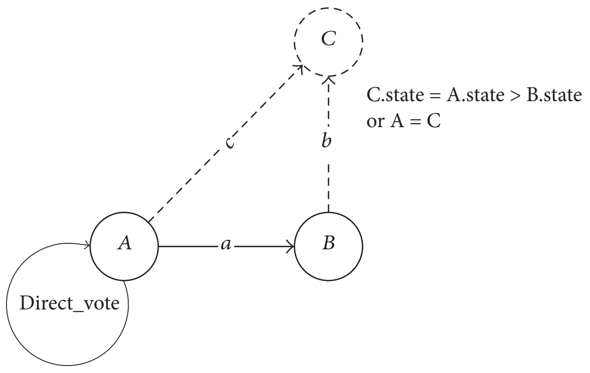


Figure 14: $C.state = A.state > B.state$ or $A = C$.

the distribution of the direct vote value and the indirect vote value can be determined. The equations for the distribution of votes' values are shown by

$$\begin{gathered} direct\_vote = 0 \\ indirect\_vote = k. \end{gathered} \tag{11}$$

If node C is not in $A.Q\_max\_k\_heap$, this situation is shown in Figure 13. It indicates that node A hardly or never contacts node C. The equations for the distribution of votes' values are shown by

$$\begin{gathered} direct\_vote = 0 \\ indirect\_vote = 0. \end{gathered} \tag{12}$$

Subscenario 3.2 ($C.state = A.state$). According to the relationship between node A and node C, it can be classified into two situations as well. It is shown in Figure 14.

(i) $B.adj\_max\_node = A$. The *state* value of node A is the greatest in the neighborhood; that is, $A.state = B.adj\_max\_state$.

(ii) $B.adj\_max\_node \neq A$. The *state* value of node A is the same as that of node C, which means $C.state = A.state = B.adj\_max\_state$.

The distribution strategies of vote value are the same. When the total value of votes k is decided, node A only adds the direct vote to itself rather than votes for node B or node C. So the equations for the distribution of votes' values are shown by

$$\begin{gathered} direct\_vote = k \\ indirect\_vote = 0. \end{gathered} \tag{13}$$

The proposed algorithms used in the voting process are shown by the following pseudocode in detail. The voting process occurs in node A which sends the votes as shown in Algorithm 1 and the voting process occurs in the node B which receives the votes as shown in Algorithm 2.

3.4. Updating Process

3.4. Updating Process. Each node repeats the updating process at a predefined time interval *INTERVAL_TIME* independently, during which each node updates its own node state information based on the information obtained from other nodes. In the updating process, nodes will update the action probability vectors and determine whether to set themselves as caching nodes. The updating process fundamentally depends on the learning automaton assigned to each node.

As mentioned, *dor_prob* represents the action probability of setting a node as a caching node. Each learning automaton updates the action probability *dor_prob* following the learning automata mechanism and the reinforcement signal β. Then the state of every node is determined by the action probability *dor_prob* and the predefined threshold value of action probability *DOR_THRESHOLD*. The updating process is shown as follows.

First of all, the node suspends all contacts with other nodes before starting an updating process. Based on the neighboring nodes state information recorded in *M_adj_node*, the node calculates the average *state* value of neighboring caching nodes *avg_adj_dor* and the average *state* value of neighboring noncaching nodes *avg_adj_dee*. Meanwhile, nodes also get informed of whether a neighboring node is a caching node according to the *is_dominator* value stored in *M_adj_node*.

The reinforcement signal β is shown by (14), where *sum_adj_dee* represents the number of neighboring noncaching nodes. If the node gets the votes (state > 0) and is surrounded by noncaching nodes, it is rewarded because it may be a good caching candidate; otherwise it is penalized since it is surrounded by a caching node already.

$$\beta(n) = \begin{cases} 1 & state \neq 0 \text{ and } sum\_adj\_dee > 1 \\ 0 & otherwise. \end{cases} \tag{14}$$

According to the reinforcement signals, nodes start to update the action probability *dor_prob*. All learning automata reward the action if $\beta = 1$ whereas they penalize the action if $\beta = 0$. Let *dor_prob*(n) be the action probability at instant n.

```
Require: Node $A$ makes a contact with node $B$.
Ensure: Node $A$ determines vote values and the vote-receiving node $C$
(1) procedure VOTING PROCESS($id_{\text{node}B}$)
(2)     $A.M\_contacts\_rec[B] \leftarrow A.M\_contacts\_rec[B] + 1$
(3)     $a \leftarrow A.M\_contacts\_rec[B]$
(4)     Insert node $B$ into $A.Q\_max\_k\_heap$
(5)     if node $B$ in $A.Q\_max\_k\_heap[1..VOTE\_K]$ then
(6)         Let $index$ denote the position of node $B$ placed in $A.Q\_max\_k\_heap$
(7)         if the relationship between $A.state$ and $B.state$ is uncertain then
(8)             $direct\_vote \leftarrow Q\_vote\,[index]$
(9)             $indreict\_vote \leftarrow 0$
(10)        else if node $A$ contains the node information of node $B$ then
(11)            $b \leftarrow B.M\_contacts\_rec\,[C]$
(12)            if $A.state < B.state$ then                          ▷ $A.state < B.state$
(13)                if $B.state = C.state$ then                      ▷ $B.state = C.state$
(14)                    $direct\_vote \leftarrow Q\_vote\,[index]$
(15)                    $indreict\_vote \leftarrow 0$
(16)                else                                             ▷ $B.state < C.state$
(17)                    if node $C$ in $A.Q\_max\_k\_heap[1..VOTE\_K]$ then
(18)                        $c \leftarrow A.M\_contacts\_rec[C]$
(19)                        if $a > b$ then
(20)                            $direct\_vote \leftarrow \lfloor Q\_vote[index] \times a/(a+c) * (a-b)/a \rfloor$
(21)                        else
(22)                            $direct\_vote \leftarrow 0$
(23)                        end if
(24)                        $indirect\_vote \leftarrow Q\_vote[index] - direct\_vote$
(25)                    else if node $C$ not in $A.Q\_max\_k\_heap[1..VOTE\_K]$ then
(26)                        if $a > b$ then
(27)                            $direct\_vote \leftarrow \lfloor Q\_vote[index] \times (a-b)/a \rfloor$
(28)                        else
(29)                            $direct\_vote \leftarrow 0$
(30)                        end if
(31)                        $indirect\_vote \leftarrow Q\_vote[index] - direct\_vote$
(32)                    end if
(33)                end if
(34)            else if $A.state > B.state$ then                     ▷ $A.state > B.state$
(35)                if $A.state = C.state$ then                      ▷ $A.state = C.state$
(36)                    $A.state \leftarrow A.state + Q\_vote\,[index]$
(37)                    $direct\_vote \leftarrow 0$
(38)                    $indirect\_vote \leftarrow 0$
(39)                else                                             ▷ $A.state > C.state$
(40)                    if node $C$ in $A.Q\_max\_k\_heap[1..VOTE\_K]$ then
(41)                        $direct\_vote \leftarrow 0$
(42)                        $indirect\_vote \leftarrow Q\_vote\,[index]$
(43)                    end if
(44)                end if
(45)            end if
(46)        end if
(47)    end if
(48)    Generate and transmit direct vote and indirect vote message to node $B$
(49)    Generate and transmit node information of node $A$ to node $B$
(50) end procedure
```

ALGORITHM 1: The voting process in the vote-sending node.

When the action is rewarded, the recurrence equation is shown by

$$dor\_prob\,(n+1) = dor\_prob\,(n) + p \cdot \left[1 - dor\_prob\,(n)\right]. \tag{15}$$

And when the action is penalized, the recurrence equation is shown by

$$dor\_prob\,(n+1) = p \cdot dor\_prob\,(n). \tag{16}$$

```
Require: Node $B$ receives votes and node information messages from node $A$.
Ensure: Node $B$ updates its node information and records node information of node $A$
(1) procedure VOTING PROCESS($direct\_vote, indirect\_vote, A.node\_information$)
(2)     $B.state \leftarrow B.state + direct\_vote$
(3)     Store $A.node\_information$ in $B.M\_adj\_node$
(4)     $M\_indirect\_node[C] \longleftarrow M\_indirect\_node[C] + indirect\_vote$
(5)     Update $adj\_max\_node$
(6) end procedure
```

ALGORITHM 2: The voting process in the vote-receiving node.

p denotes the reward and penalty parameter and determines the amount of increases and decreases of the action probabilities. p is shown by

$$p = \begin{cases} \min\left(1, \dfrac{state \times (sum\_adj\_dee + 1)}{state + sum\_adj\_dor \times avg\_adj\_dor}\right) & state \neq 0 \\ 0 & state = 0. \end{cases} \quad (17)$$

and $0 \leq p \leq 1$.

If the state of a node is 0, this node does not get any vote during the last updating period. In this case, the node cannot get the reward by making the value of *dor_prob* same as before. If penalized the value of *dor_prob* is set to 0. So the possibility of becoming a CNS node will be reduced significantly. Otherwise, this node does get some votes during the last updating period, and the reward is decided by the ratio of its state value to the maximum state value of its neighbor. It means the node that gets the most votes among its neighbors will be rewarded most.

The node state is determined by *dor_prob* and *DOR_THRESHOLD*. If $dor\_prob \geq DOR\_THRESHOLD$, the node is set as a caching node; otherwise the node is set as a noncaching node. It is different with the classical learning automata selecting its action according to its action probability vector. Therefore, the nodes that are not suitable for caching data get no chance to become CNS nodes. It leads to significant performance loss, especially in the beginning period and the period when the node contacting pattern changes. To address this issue, we modified the classical learning automata to our version to make only suitable nodes become CNS node. It may cause suboptimal solution. However, it can find a reasonable solution quickly and may be more appropriate for dynamic DTNs.

Finally, the node removes the stored node information items whose lifetime has expired from the buffer. When the node state is determined and the obsolete information is eliminated, the node resumes working and contacting other nodes. Then the updating process is finished. And CNS is comprised of all the caching nodes in the network and might change along with the network operating. More details about the updating process are shown by the pseudocode in Algorithm 3.

4. Cache Based Data Access

In this section, we focus on data distribution and cache replacement in the network.

4.1. Data Transmission. There are several different routing algorithms in DTNs. Each algorithm has its own advantages and disadvantages. The major concern of all algorithms is to balance the trade-off between data delivery rates and transmission overhead, and the appropriate algorithm to use depends on the given circumstances.

Epidemic routing [8] now is widely used in wireless networks. Epidemic routing is a flooding-based algorithm, as nodes continuously replicate and forward data to newly discovered nearby nodes indiscriminately. However, epidemic routing is a resource-hungry algorithm because it fails to reduce redundant and unnecessary data replications. Thus further techniques and protocols are proposed to improve the performance of data transmission in DTNs.

Generally, there are mainly two types of data transmission in our proposed scheme as follows:

(i) *Data requests*: when a node requests some data, it only multicasts the queries to nearby caching nodes. And the caching nodes reply to the node with the data if they cache the data copy.

(ii) *Data distribution*: when a new data item is generated by a random node, the data needs to be distributed to all the caching nodes. One copy of data is cached at each caching node.

To achieve effective data transmission in DTNs, we need to find an appropriate routing protocol. Considering the lack of connectivity and instantaneous end-to-end paths in DTNs, popular ad hoc routing protocols such as AODV and DSR fail to establish routes. To overcome these challenges, we employ the PRoPHET routing protocol [9], which is specifically designed to solve routing problems in DTNs. In the adaptive algorithm, each node M in DTNs maintains a delivery predictabilities vector. Each delivery predictability $P(M, D)$ indicates the probability for successful delivery from node M to the destination node D. The delivery predictabilities are determined by the following rules [9].

(i) When a node M makes a contact with another node E, the rule is shown by

$$P(M,E)_{\text{new}} = P(M,E)_{\text{old}} + \left(1 - P(M,E)_{\text{old}}\right) \times L \qquad 0 < L < 1, \quad (18)$$

where L is a constant parameter.

```
Require: Each node updates the action probability vector and eliminates obsolete votes.
Ensure: Each node updates the action probability vector and construct a new caching node set.
(1) procedure UPDATING PROCESS
(2)     ∇ Update node information and eliminate obsolete information
(3)     Eliminate the obsolete vote weight values from state
(4)     Eliminate the obsolete node information from M_adj_node
(5)     Eliminate the obsolete indirect_vote from M_indirect_vote
(6)     ∇ Determine the caching nodes set using the learning automata
(7)     if $state \neq 0$ then                ▷ determine the reward and penalty parameter $p$
(8)         $p = \min(1, state \times (sum\_adj\_dee + 1)/(state + sum\_adj\_dor \times avg\_adj\_dor))$
(9)     else
(10)        $p = 0$
(11)    end if
(12)    if $state \neq 0$ and $sum\_adj\_dee > 1$ then      ▷ determine the reinforcement signal $\beta$
(13)        $\beta = 1$
(14)    else
(15)        $\beta = 0$
(16)    end if
(17)    if $\beta = 1$ then                   ▷ Reward
(18)        $dor\_prob = dor\_prob + p \times (1 - dor\_prob)$
(19)    else if $\beta = 0$ then              ▷ Penalty
(20)        $dor\_prob = p \times dor\_prob$
(21)    end if
(22)    if $dor\_prob \geq DOR\_THRESHOLD$ then   ▷ Determine the caching node
(23)        $is\_dominator = true$
(24)    else
(25)        $is\_dominator = false$
(26)    end if
(27) end procedure
```

ALGORITHM 3: The updating process.

(ii) The delivery predictabilities for all destination nodes like node D decay along with the time; the rule is shown by

$$P(M, D)_{\text{new}} = P(M, D)_{\text{old}} \times \gamma^{K} \quad 0 < \gamma < 1, \tag{19}$$

where γ is the decaying constant parameter and K is the elapsing time slots since the last decaying.

(iii) When exchanging delivery predictabilities vectors between node M and node E, the delivery predictability of destinations D is updated based on the transitive property of predictability for which node E has a $P(E, D)$ value on the assumption that M is likely to meet node E again. The rule is shown by

$$\begin{aligned} P(M, D)_{\text{new}} = {} & P(M, D)_{\text{old}} + \left(1 - P(M, D)_{\text{old}}\right) \\ & \times P(M, E) \times P(E, D) \times \beta, \end{aligned} \tag{20}$$

where β is a scaling constant.

In the proposed scheme, all nodes are either caching nodes or adjacent to caching nodes within a short probabilistic distance so that most data requests can be replied quickly. Due to the distribution pattern of caching nodes in DTNs, the data are supposed to reach destination nodes within a few hops. On the other hand, data distribution in our proposed scheme is flooding-based in nature. When new data are generated by a random node, the node merely forwards the data to the neighboring known caching nodes. When a caching node receives the newly generated data, it caches the data and then forwards the data to other neighboring known caching nodes. Caching nodes receiving the data repeat the process until each caching node caches a data copy.

To improve routing performance and reduce the waste of resources in networks, we also introduce some rules and restrictions into the routing protocol. First, we set a hop limit as h indicating that a data copy will be discarded after a h-hop delivery. An appropriate hop limit h is used to eliminate redundant data transmitted in networks and prevents routing overflooding effects. Additionally, not only does the asymmetric probability distribution in PRoPHET protocol prevent data from being transmitted in loop to some degree, but some transmitted data copies trapped in loop will perish when exceeding the hop limit. It avoids useless repetitive data transmission and saves resources in networks.

We also enhance PRoPHET routing protocol by proposing the predefined probability threshold $P_{\text{threshold}}$ and a data delivery rule. When node M needs to transmit the data to the destination node D through the relay node E, nodes would first check the relationship among $P(M,D)$ and $P(E,D)$ and the probability threshold $P_{\text{threshold}}$. If $P(E,D) < P_{\text{threshold}}$ and $P(E,D) < P(M,D)$, the data copy would not be delivered. This rule avoids data transmission through some paths of low transmission successful rate and improves routing performance in networks.

4.2. Data Replacement. One major restriction for any caching scheme is the limited storage space in caching nodes. Cache replacement is consequently necessary when caching buffers run out, where some obsolete and less popular data will be removed so that new data can be cached. The latency and the hit ratio are two major concerns for cache replacement. The cache replacement strategy used in the scheme is designed to optimize the trade-off between the delay and data accessibility. In the proposed scheme, cache replacement occurs under two circumstances.

(i) When newly generated data reach a caching node lacking enough space for storage, the caching node removes obsolete data to cache new data. The data copies cached in caching nodes are ordered by utility values. The caching node will discard the data of lowest utility values until there is enough space for the new data.

(ii) When a former caching node ceases to be a caching node, the data copies cached in the node will be removed and transmitted to other caching nodes if necessary. The node strives to forward the data with the utility values from high to low to other neighboring caching nodes and removes the data cached in itself.

Of all different cache replacement strategies, utility-based cache replacement strategy is most DTNs-based and widely used by many researches and applications [10–12]. We use utility-based cache replacement strategy in our scheme to address limited memories in caching nodes.

In the proposed scheme, utility value determined by utility functions is assigned to each data copy cached in caching nodes. Utility values indicate the frequency of each data being requested based on the query history. The occurrence of data requests follows a Poisson distribution [4, 13]. We assume that there are k requests to the data in the time period $[t_1, t_k]$, then the parameter for Poisson distribution is $\lambda_d = k/(t_k - t_1)$ and the utility value for each data decays along with time as well. At time t, the utility value w_i for data i is determined by the utility function $w_i = 1 - e^{-\lambda_d \cdot (t-t_k)}$.

5. Performance Evaluation

In this section, we conduct simulations to evaluate the performance of the proposed caching scheme and compare the merits and demerits with two existing data caching schemes, Intentional Cache [4, 13] and Adaptive Cache [14]. Each simulation is repeated multiple times for statistical convergence. The following metrics are used for evaluations:

(i) Successful ratio, the ratio of forwarding the requested data to requesters successfully.

(ii) Data access delay, the average delay for responding to queries with the requested data.

(iii) Number of caching nodes, the number of nodes set as caching nodes in a certain period. The number of caching nodes to some extent reflects the caching overhead and the number of cached data copies in the network.

5.1. Simulation Setting. The performance evaluations are performed on the *Infocom06* trace, which is collected by the Haggle project [15], and *MIT Reality* trace, which is collected by the MIT Reality mining project [16]. In *Infocom06* trace, there are 100 participants with iMote devices to record their contacts in 4 days. *Infocom06* trace contains 227657 internal contacts. In *MIT Reality* trace, there are 97 participants with cellphones to record their contacts in 246 days. It records 114046 internal contacts.

Unlike the evaluation simulations conducted in [14] and [4, 13] where the first half of the trace is used as the warm-up period and only the second half of trace is used for the performance evaluation, the data and queries in our simulations are generated throughout the whole trace.

Each node periodically generates new data, and the probability of generating new data is p_G which is set as 0.2. Like the simulations in [4, 13], each generated data has a finite lifetime which is set as T, and the period for data generation decision is also set as T.

The queries are randomly generated at all nodes. Similar to the query pattern used in [4, 13], the query pattern follows Zipf distribution [17]. We assume $P_j \in [0,1]$ as the probability that data j is requested and M as the number of data items in the network. Then we have $P_j = (1/j^s)/(\sum_{i=1}^{M}(1/i^s))$ where s is an exponent parameter. At intervals of $T/2$, each node determines whether to request data j with the probability P_j.

In [4, 13], the caching performance of Intentional Caching is evaluated by compared with several other different caching schemes, including No Cache, Random Cache, CacheData, and Bundle Cache. The results show that Intentional Caching proposed in [4, 13] has overall advantages over other caching schemes. This is the reason that we choose Intentional Caching scheme as a comparing object. Similar to our schema, Adaptive Cache uses learning automata to decide whether a node should be caching node or no. Hence, we select Adaptive Cache as the other comparing object.

5.2. Caching Performance. In [4, 13], the caching performance of Intentional Caching scheme is only evaluated on the *MIT Reality* trace. Consequently, to compare our caching scheme with Intentional Caching scheme, we also only use the *MIT Reality* trace to evaluate the caching performance of our proposed scheme. In Intentional Caching, the number of NCLs used to cache data is set as 8, while the number of caching nodes in our caching scheme varies according to

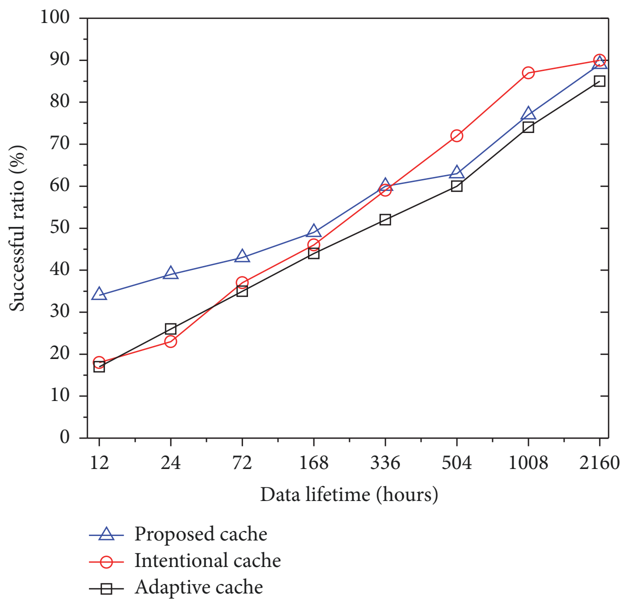


Figure 15: Successful ratio of data access with different data lifetime.

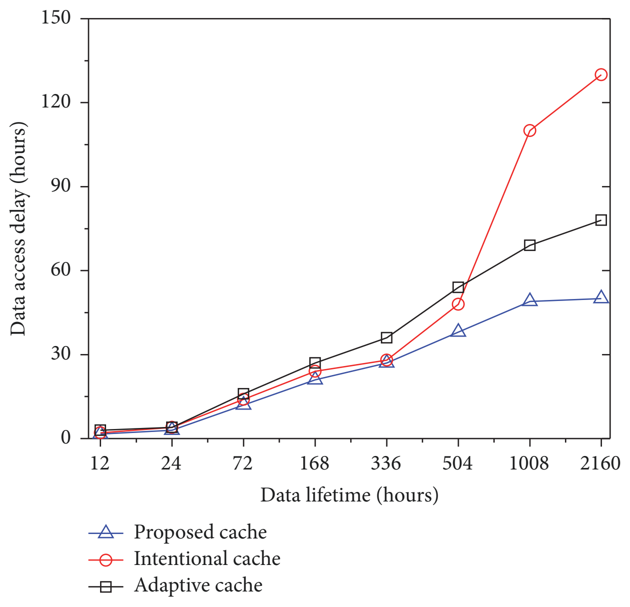


Figure 16: Data access delay with different data lifetime.

the contact history and network conditions. And we set the *DOR_THRESHOLD* as 0.8.

We compare the performance of these two caching schemes with different T shown in Figures 15 and 16. When T increases from 12 hours to 2160 hours, the successful ratio of both caching schemes is improved. While the Intentional Caching scheme achieves a satisfactory successful ratio and short delay of data access, our proposed caching scheme provides a similar performance in terms of successful ratio and reduces the data access delay significantly.

As shown in Figure 15, our scheme displays a better successful ratio with short data lifetime, while the performance of Intentional Caching slightly outmatches our scheme when T increases. On the other hand, our scheme reduces data access delay significantly as shown in Figure 16. The access delay for both caching schemes remains relatively low when T is small. But as T increases, the delay for Intentional Caching increases more significantly while our scheme achieves 60% shorter delay than Intentional Caching.

The Intentional Caching scheme uses global network information to select NCL (network central location) nodes. NCL nodes are the nodes that have the most frequent contact with the rest network nodes. Once selected, NCL nodes cache the data in the whole time. Without using the global network information, our scheme is distributed and makes it possible for nodes to decide whether to be caching nodes themselves. The CNS is reconstructed occasionally to respond to the changes in networks to ensure that most data requests can be replied more quickly; thus our scheme performs over Intentional Caching given short data lifetime. It also explains the high successful ratio of Intentional Caching given long data lifetime.

If the data are not found in NCL, then the request is broadcasted to the entire network until the data source is found. The storage is limited, if the data lifetime increases; there may be no enough space in the NCL nodes to cache all the data items. It will cause performance degrading such as longer delay. In our scheme, the node uses the local information, the nodes, and their roles it contacted directly or indirectly to decide whether it becomes a caching node or not. It may not get a global optimal solution, but more nodes get the chance to cache the data. For example, a node making few contacts with the other nodes may become a CNS node if all its contacted nodes are not CNS nodes. The storage limitation has less effect on the performance of our scheme comparing with the Intentional Caching.

Comparing with our scheme, the performance of the Adaptive Caching scheme has a similar variation tendency when the data lifetime varies since these two schemes both use learning automata. However, the performance is lower than our scheme since it uses forwarding ratio of the node as the indicator of the network environment, which make it highly dependent on the data traffic pattern and underlying routing protocols. For example, if there is no previous traffic in a certain area, all the nodes in this area can not get the chance to become the caching nodes. If a node in this area wants to access a certain data, it may need long delay since no nearby nodes cache the requested data.

In our scheme, the size of CNS varies along with the time, which may need more caching nodes than NCLs sometimes. The selection of caching nodes contributes to the difference and details will be discussed in the next section.

We also evaluated data access performance with different average data size when T is set as 168 hours. The results are shown in Figures 17 and 18. When the average data size increases, the performance of Intentional Caching weakens, while the performance of our proposed scheme is less susceptible to the variation in data size. When the average data size increases from 20 Mb to 200 Mb, the successful ratio of Intentional Caching decreases from 60% to 45%, but the

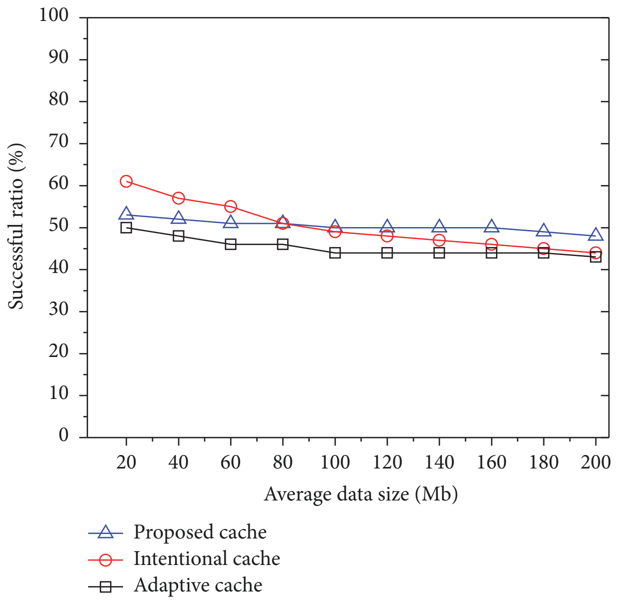


Figure 17: Successful ratio with different data sizes.

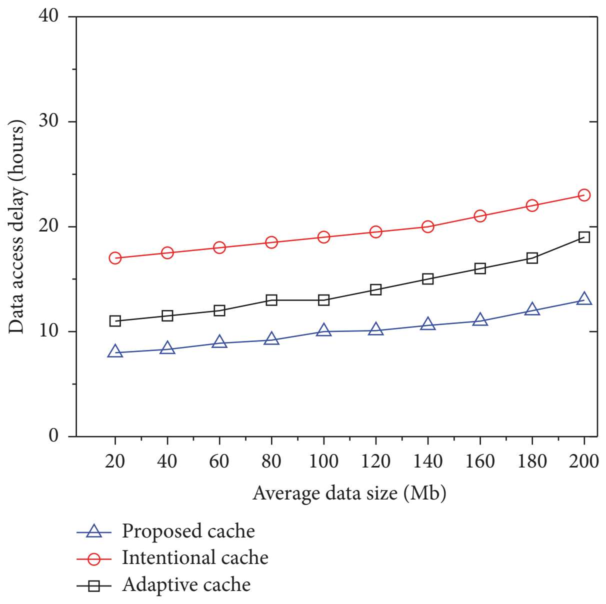


Figure 18: Data access delay with different data sizes.

successful ratio of our proposed caching scheme remains around 50%. And the data access delay of our proposed scheme is overall shorter than that of Intentional Caching. As we discussed above, when the data item size is large, the storage limitation causes the performance of the Intentional Caching degrading. In our scheme, more nodes get the chance to cache the data, leading to a better performance. The Adaptive Caching scheme is also not as sensitive to the variation of data size as the Intentional Caching scheme since it is also a distributed and adaptive scheme. The performance is lower than our scheme since it uses forwarding ratio as we discussed before.

Comparatively, all caching schemes adopt effective cache replacement strategies to cache the most appropriate data. But the intelligent distribution of caching nodes in our scheme ensures that most nodes are able to contact multiple caching nodes in shorter opportunistic distances, which increases the successful ratio and reduces data access delay significantly.

5.3. Selection of Caching Nodes. When determining the caching nodes in the networks, we are inclined to reduce the number of caching nodes so as to lower the overhead while maintain the overall performance. In Intentional Caching, the number of NCLs is predefined before simulations. Using *Infocom06* trace, they set $T = 3$ hours and evaluate the impact of the numbers of NCLs on the data access performance on *Infocom06* trace. The result shows that $K = 5$ is the best choice for the *Infocom06* trace. But there exist some defects in the selection of NCLs. The probabilistic selection metric is in fact based on the global network state information, which is hard to obtain in practice. Moreover, a network warm-up period is required to build up the NCLs.

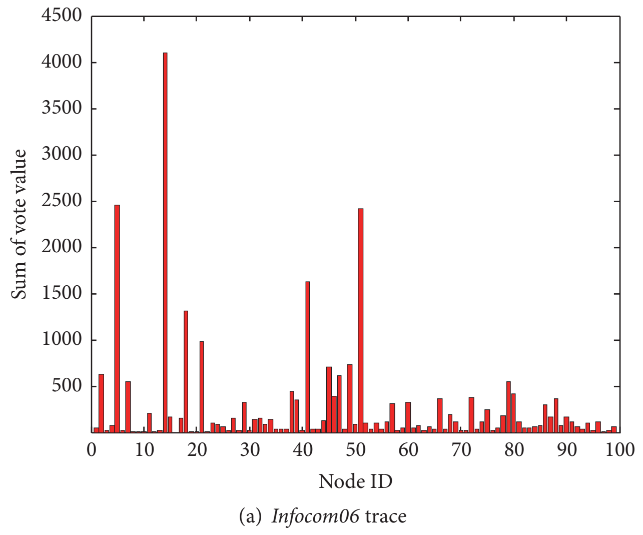
As for our proposed caching scheme, the selection of caching nodes is completely based on local network state information and avoids the warm-up period. The number of caching nodes, instead of being constant, varies along with the changing networks environment. In this section, we study the major factors influencing the number of caching nodes.

The sum of vote values for each node serves as an important metric for the selection of caching nodes. The proposed vote mechanism ensures the differentiation of the sums of vote value. We evaluate the sum of vote values for each node using *Infocom06* trace and *MIT Reality* trace. T is set as 14 hours for *Infocom06* trace and 7 days for *MIT Reality* trace.

The results in Figure 19 show that the sum of vote values for each node is highly skewed in each trace. The obvious differentiation is conducive to the selection of caching nodes and supports the validity of our proposed caching scheme. And it also indicates that the selected caching nodes can be easily accessed by other nodes.

Although the number of caching nodes varies over time due to the proposed mechanism, the distribution of the metrics is still consistent with the selection of caching nodes to some extent. For instance, the average number of caching nodes is around 10 when T is 14 hours in *Infocom06* trace. As shown in Figure 19(a), the number of caching nodes is generally no more than 10 and we can easily select 6 nodes as caching nodes of greater vote value sums. And the nodes we select from Figure 19(a) all serve as the primary caching nodes in the trace.

Then, we evaluate the impact on the number of caching nodes by using *Infocom06* trace and *MIT Reality* trace. Considering the lack of persistent connection among nodes, in a certain period, some nodes in the network may be unable to contact any caching nodes or even any other nodes at all. During the simulation, we record the number of caching nodes in the network every T hours; meanwhile each noncaching node also periodically finds out whether it can contact a caching node successfully. We assume the nodes which are not caching nodes as well as unable to contact any



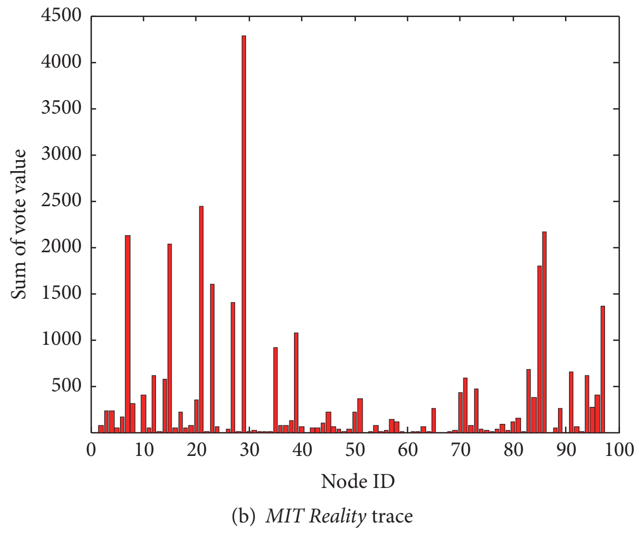
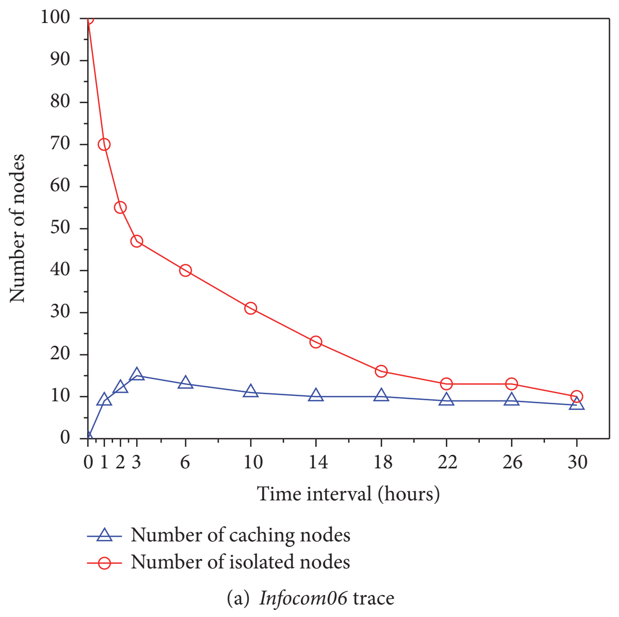


Figure 19: Sums of vote values for each node on realistic traces.



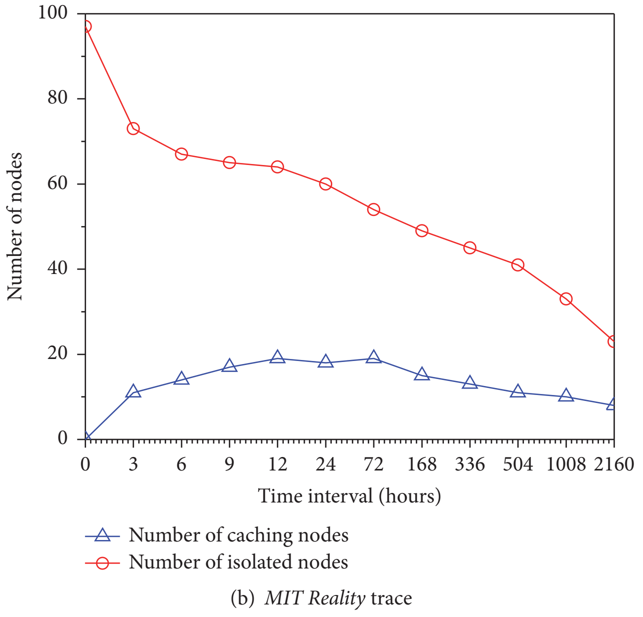


Figure 20: The number of caching nodes and isolated nodes.

other caching nodes as isolated nodes. As shown in Figure 20, the number of caching nodes is reduced when T increases. Generally, caching nodes only make up 5% to 15% of all the nodes. On the other side, the number of isolated nodes decreases significantly when T increases. In Figure 20(a), there are over 40 isolated nodes when $T = 3$ hours while there are only 5 isolated nodes when $T = 30$ hours. However, there are more isolated nodes in Figure 20(b), which is largely caused by the contacts pattern among nodes in *MIT Reality* trace. Although some nodes are still isolated inevitably, the reduction of isolated nodes still improves the overall performance.

The change of the number of caching nodes also can be seen as a clear indicator reflecting the converging characteristics of our scheme. From Figure 20, the number of caching nodes becomes relatively stable quickly, which means learning automata reaches a stationary status.

However, the number of caching nodes is not the only concern. The effectiveness of caching nodes requires that the states of caching nodes remain relatively constant; that is, when a node sets itself as a caching node, it is necessary for the node to keep itself as a caching node for a continuous period to cache data and reply requests. Let the action probability of setting a node as a caching node be $\alpha \in [0, 1]$. Since the states of nodes are determined by the action probability vector, the values of α for some nodes remain comparatively constant for a long and continuous period while others might vary and fluctuate dramatically which are likely to disturb the stability of the network. The variation trends of the action probability reveal the frequency of nodes changing their

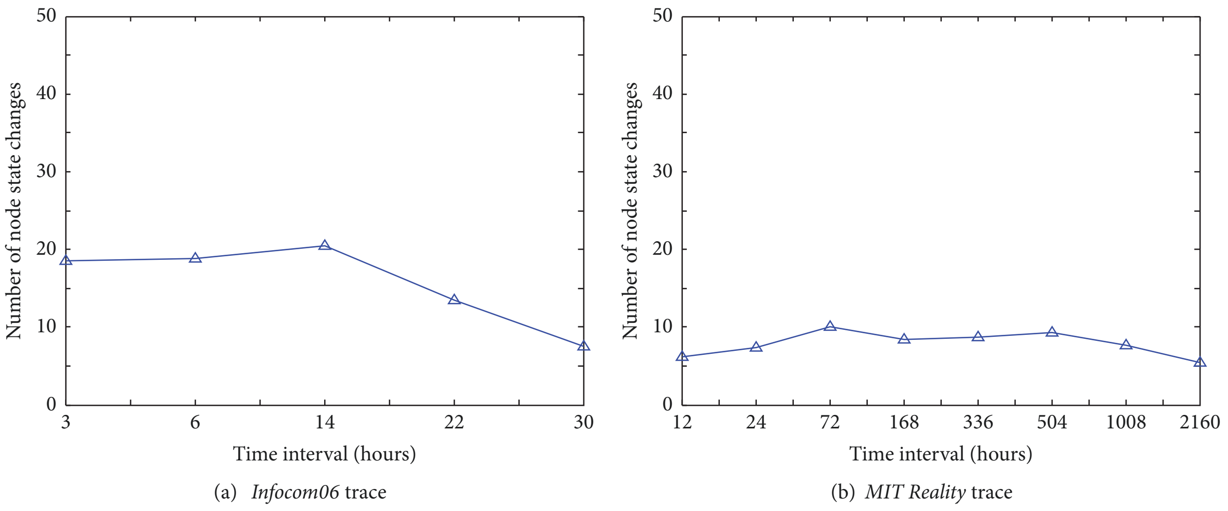


Figure 21: The frequency of node states change.

states either from caching nodes to noncaching nodes or the other way around. The average frequency of states change for all nodes is shown in Figure 21. The frequency of states change remains comparatively constant, which also shows our scheme converges quickly. *Infocom06* trace describes the movement of people participating in an international conference, and *MIT Reality* trace describes the movement of the paper studying and working in the same lab. The former case has the higher dynamics. Therefore, the frequency of node state changing in *Infocom06* trace is overall higher than that in *MIT Reality* trace.

5.4. Summary of Simulation. We compare the performance of our proposed caching scheme with the Intentional Caching scheme [4] and the Adaptive Caching scheme [14]. The successful ratios of data access and data access delay are evaluated. Our proposed caching scheme exhibits a similar overall performance to that of Intentional Caching under some circumstance, while showing a best performance insofar as the data access delay. However, the most noticeable advantage of our proposed scheme over other schemes is that our proposed scheme requires no global network state information to select the caching nodes and is able to adjust to the variations in network topology automatically. In conclusion, our proposed caching scheme proves to be an effective caching scheme.

6. Related Work

The general concept of Delay Tolerant Networks is proposed in 2003. Since then, a growing number of researches have drawn increasing attention to DTNs and sought to developing effective schemes to improve the overall performance of DTNs-based networks.

Issues on data transmission in DTNs derive from epidemic routing in ad hoc networks [8]. However, the "carry-and-forward" mechanism [18] used in epidemic routing is ineffective in DTNs. To improve the performance of routing in DTNs, later studies incorporate semi-Markov chains [19] and Hidden Markov Models [20, 21]. Other studies [22–24] propose routing protocols based on historic contacts records and predications of mobility patterns. Taking a step further, [25] offers a hybrid routing algorithm and explores the application for providing health services in rural areas.

To facilitate data access in DTNs, caching techniques have been extensively studied and cooperative caching schemes are designed to intentionally coordinate multiple nodes to cache and share data. Different cooperative caching strategies that apply to DTNs are proposed. For instance, [26] provides a cooperative caching scheme based on social relationships among nodes and [27] proposed a content floating-based cooperative caching strategy. However, most caching schemes fail to address the issue on the changing network topology or the lack of global network state information in DTNs; thus these schemes rarely remain a consistently high performance in a long term.

To overcome the variability of networks topology, cellular automata [28, 29] are introduced into traditional wireless networks. Given the limited resource supplied to nodes in wireless networks, [30] provides an energy-conservation solution to maximize the life time of the network by minimizing the energy consumption based on cellular automata. Then on the basis of the cellular automata, later studies seek to develop self-organized and self-adaptive network schemes. Enlightened by the solutions [31–33] to graph theory problems using learning automata mechanism, distributed learning automata are introduced to wireless network clustering algorithms [34], scheduling methods [35], and routing protocols [36]. Our scheme takes the advantage of the learning automata theory to overcome the unpredictable and inconsistent network environment in DTNs.

Hamid and Meybodi [37, 38] proposed a continuous action-set learning automaton based call admission control algorithm to minimize the blocking probability of the new calls subject to the constraint on the dropping probability of the handoff calls in cellular mobile networks and theoretically prove that it converges to the optimal solution. We use the same idea to the caching based data access in DTNs. Since we can not model the data access process as a random process like Markov process, we use the simulation to demonstrate the effectiveness of out method.

When evaluating the performance, besides [4], we also compare our scheme with other works theoretically. In [13], authors make an improvement for the scheme proposed in [4] and the improved scheme is able to build the set of caching nodes in the absence of global network state information. However this improvement is at the great cost of the scheme performance. The inconsistency in the selection of caching nodes in networks appears in [13], undermining the reliability of the selection of caching nodes. To neutralize the inconsistency, the broadcasting period has to be extended and additional information exchange between two contacted nodes is needed. On the other hand, some major deficits in [4] still exist in [13]. For instance, the scheme in [13] still cannot overcome the changing networks topology and needs a warm-up period to select the Caching Node Set.

Reisha Ali proposes a learning automata based scheme to choose a set of nodes for caching which could contribute more to the entire network [14]. Although it has something in common with our proposed scheme, they are essentially different. It uses forwarding ratio of the node as the indictor of the network environment, which make it highly dependent on the data traffic pattern and underlying routing protocols. Furthermore, like the algorithm in [4, 13], the algorithm in [14] needs a warm-up period for the selection of caching nodes and is unable to adapt to the dynamic changing of network topology.

7. Conclusion

In this paper, we propose a self-organized and self-adaptive caching scheme based on distributed learning automata. The basic idea of our proposed caching scheme is to maintain the Caching Node Set (CNS) which offers easy and effective data accessibility to all nodes. The selection and maintenance of CNS are achieved by the learning automata assigned to each node. Through the well-designed voting and updating processes, all nodes can cooperate spontaneously to achieve effective data access in DTNs, optimizing the overall performance of networks. More importantly, our proposed caching scheme is specially designed for the DTNs environment, which is characterized by the lack of the global network state information, the unpredictable node mobility, and high forwarding latency. The trace-driven simulations exhibit the effectiveness and advantages of our proposed scheme compared with other existing caching schemes.

8. Future Work

In our proposed distributed caching scheme, we use the utility-based cache replacement strategy to accommodate the limited storage space in each node; however, other possible and promising cache replacement strategies are uncovered in our evaluations. Cache replacement strategies have an important influence on the performance and studying the effects of different strategies on the performance of our scheme is necessary. Furthermore, cache redistribution may cause traffic overhead and have negative effects on the performance in certain periods, especially when the nodes change their states. Further research is still required in the future.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China under Grant no. 51435009 and CERNET Innovation Project under Grant no. NGII20170323.

References

[1] K. Fall, "A delay-tolerant network architecture for challenged internets," in *Proceedings of the the 2003 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp. 27–34, ACM, 2003.

[2] K. Chen and H. Shen, "DTN-FLOW: Inter-landmark data flow for high-throughput routing in DTNs," in *Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2013*, pp. 726–737, May 2013.

[3] L. Ziyi and F. Jianhua, "Delay/disruption tolerant network and its application in military communications," in *Computer Design and Applications (ICCDA), 2010 International Conference*, vol. 5, pp. V5–231, 2010.

[4] W. Gao, G. Cao, A. Iyengar, and M. Srivatsa, "Supporting cooperative caching in disruption tolerant networks," in *Proceedings of the 31st International Conference on Distributed Computing Systems, ICDCS 2011*, pp. 151–161, July 2011.

[5] Y.-K. Ip, W.-C. Lau, and O.-C. Yue, "Forwarding and replication strategies for DTN with resource constraints," in *Proceedings of the 2007 IEEE 65th Vehicular Technology Conference - VTC2007-Spring*, pp. 1260–1264, April 2007.

[6] K. S. Narendra and M. A. Thathachar, "Learning automata—a survey," *The Institute of Electrical and Electronics Engineers Systems, Man, and Cybernetics Society*, vol. SMC-4, pp. 323–334, 1974.

[7] J. Akbari Torkestani and M. R. Meybodi, "Learning automata-based algorithms for finding minimum weakly connected dominating set in stochastic graphs," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 18, no. 6, pp. 721–758, 2010.

[8] A. Vahdat and D. Becker et al., "Epidemic routing for partially connected ad hoc networks," Tech. Rep. CS-200006, Duke University, 2000.

[9] A. Lindgren, A. Doria, and O. Schelèn, "Probabilistic routing in intermittently connected networks," *Mobile Computing and Communications Review*, vol. 7, no. 3, pp. 19-20, 2003.

[10] A. Jaleel, H. H. Najaf-Abadi, S. Subramaniam, S. C. Steely Jr., and J. Emer, "CRUISE: Cache replacement and utility-aware scheduling," in *Proceedings of the 17th International Conference on Architectural Support for Programming Languages*

and Operating Systems, ASPLOS 2012, vol. 40, pp. 249–259, March 2012.

[11] N. Chand, R. C. Joshi, and M. Misra, “Cooperative caching strategy in mobile ad hoc networks based on clusters,” *Wireless Personal Communications*, vol. 43, no. 1, pp. 41–63, 2007.

[12] P.-H. Guo, Y. Yang, and X.-Y. Li, “A P2P streaming service architecture with distributed caching,” *Journal of Zhejiang University SCIENCE A*, vol. 8, no. 4, pp. 605–614, 2007.

[13] W. Gao, G. Cao, A. Iyengar, and M. Srivatsa, “Cooperative caching for efficient data access in disruption tolerant networks,” *IEEE Transactions on Mobile Computing*, vol. 13, no. 3, pp. 611–625, 2014.

[14] A. Raian and R. Rashmi Ranjan, “An adaptive caching technique using learning automata in disruption tolerant networks,” in *Proceedings of the Next Generation Mobile Apps, Services and Technologies (NGMAST)*, pp. 186–191, Eighth International Conference, 2014.

[15] J. Scott, G. Richard, J. Crowcroft, P. Hui, D. Christophe, and A. Chaintreau, “CRAWDAD data set cambridge/haggle (v. 2006-01-31),” 2006, http://crawdad.org/cambridge/haggle/.

[16] E. Nathan and S.P. Alex, “CRAWDAD data set mit/reality (v. 2005-07-01),” 2005, http://crawdad.org/mit/reality/.

[17] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, “Web caching and zipf-like distributions: Evidence and implications,” in *Proceedings of the In INFOCOM99. Eighteenth Annual Joint Conference of the Computer and Communications Societies*, vol. volume 1, pp. 126–134, 1999.

[18] J. Wu, S. Yang, and F. Dai, “Logarithmic store-carry-forward routing in mobile ad hoc networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 735–748, 2007.

[19] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, “Efficient routing in intermittently connected mobile networks: the single-copy case,” *IEEE/ACM Transactions on Networking*, vol. 16, no. 1, pp. 63–76, 2008.

[20] W. Gao and G. Cao, “Fine-grained mobility characterization: Steady and transient state behaviors,” in *Proceedings of the 11th ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc 2010*, pp. 61–70, September 2010.

[21] A. Picu and T. Spyropoulos, “DTN-Meteo: forecasting the performance of DTN protocols under heterogeneous mobility,” *IEEE/ACM Transactions on Networking*, vol. 23, no. 2, pp. 587–602, 2015.

[22] J. Leguay, T. Friedman, and V. Conan, “DTN routing in a mobility pattern space,” in *Proceedings of the ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN ’05)*, pp. 276–283, ACM, Philadelphia, Pa, USA, August 2005.

[23] L. Jeremie, T. Friedman, and V. Conan, “Evaluating mobility pattern space routing for dtns,” Article ID 0511102, 2005, https://arxiv.org/abs/cs/0511102.

[24] A. Balasubramanian, B. Levine, and A. Venkataramani, “DTN routing as a resource allocation problem,” in *Proceedings of the ACM SIGCOMM 2007: Conference on Computer Communications*, pp. 373–384, August 2007.

[25] R. Johari, N. Gupta, and S. Aneja, “POSOP routing algorithm: A DTN routing scheme for information connectivity of health centres in hilly state of North India,” *International Journal of Distributed Sensor Networks*, vol. 2015, Article ID 376861, 2015.

[26] T. Le, Y. Lu, and M. Gerla, “Social caching and content retrieval in Disruption Tolerant Networks (DTNs),” in *Proceedings of the 2015 International Conference on Computing, Networking and Communications, ICNC 2015*, pp. 905–910, February 2015.

[27] S. Zhang, J. Wu, Z. Qian, and S. Lu, “MobiCache: Cellular traffic offloading leveraging cooperative caching in mobile social networks,” *Computer Networks*, vol. 83, pp. 184–198, 2015.

[28] R. O. Cunha, A. P. Silva, A. A. F. Loreiro, and L. B. Ruiz, “Simulating large wireless sensor networks using cellular automata,” in *Proceedings of the 38th Annual Simulation Symposium, ANSS-38*, pp. 323–330, IEEE Computer Society, April 2005.

[29] S. Athanassopoulos, C. Kaklamanis, P. Katsikouli, and E. Papaioannou, “Cellular automata for topology control in wireless sensor networks,” in *Proceedings of the 2012 16th IEEE Mediterranean Electrotechnical Conference, MELECON 2012*, pp. 212–215, March 2012.

[30] S. Adabi, A. K. Zadeh, A. Dana, and S. Adabi, “Cellular automata based method for energy conservation solution in wireless sensor network,” in *Proceedings of the 2008 International Conference on Wireless Communications, Networking and Mobile Computing, WiCOM 2008*, October 2008.

[31] J. A. Torkestani and M. R. Meybodi, “Approximating the minimum connected dominating set in stochastic graphs based on learning automata,” in *Proceedings of the 2009 International Conference on Information Management and Engineering, ICIME 2009*, pp. 672–676, April 2009.

[32] H. Beigy and M. R. Meybodi, “Utilizing distributed learning automata to solve stochastic shortest path problems,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 14, no. 5, pp. 591–615, 2006.

[33] Y. P. Chen and A. L. Liestman, “Approximating minimum size weakly-connected dominating sets for clustering mobile ad hoc networks,” in *Proceedings of the MOBIHOC 2002: PROCEEDINGS OF The Third ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pp. 165–172, June 2002.

[34] M. Esnaashari and M. R. Meybodi, “A cellular learning automata based clustering algorithm for wireless sensor networks,” *Sensor Letters*, vol. 6, no. 5, pp. 723–735, 2008.

[35] M. Jahanshahi, M. R. Meybodi, and M. Dehghan, “Cellular learning automata based scheduling method for wireless sensor networks,” in *Proceedings of the 2009 14th International CSI Computer Conference, CSICC 2009*, pp. 646–651, October 2009.

[36] A. H. F. Navid and H. H. S. Javadi, “ICLEAR: energy aware routing protocol for wsn using irregular cellular learning automata,” in *Proceedings of the 2009 IEEE Symposium on Industrial Electronics and Applications, ISIEA 2009*, pp. 463–468, October 2009.

[37] B. Hamid and M. Meybodi, *Call Admission Control in Cellular Mobile Networks: A Learning Automata Approach*, Springer, Berlin, Germany, 2002.

[38] B. Hamid and M. Meybodi, “An adaptive call admission algorithm for cellular networks,” *Computers and Electrical Engineering*, vol. 31, no. 2, pp. 132–151, March 2005.