ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.11, November 2008, pp.3042−3052
DOI: 10.3724/SP.J.1001.2008.03042

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

MANET中基于簇的缓存一致性维护策略*

谢高岗[1+], 李振宇[1,2], 陈嘉宁[1]

[1](中国科学院 计算技术研究所,北京 100190)

[2](中国科学院 研究生院,北京 100049)

Cluster-Based Consistency Scheme of Cooperative Caching in Mobile Ad Hoc Networks

XIE Gao-Gang[1+], LI Zhen-Yu[1,2], CHEN Jia-Ning[1]

[1](Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100190, China)

[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: E-mail: xie@ict.ac.cn

Xie GG, Li ZY, Chen JN. Cluster-Based consistency scheme of cooperative caching in mobile ad hoc networks. *Journal of Software*, 2008,19(11):3042−3052. http://www.jos.org.cn/1000-9825/19/3042.htm

Abstract: Cooperative caching has been adequately addressed in MANETs for QoS and cooperative computing. This paper presents a Cluster-based Consistency Scheme, CCS. In CCS, the close nodes in locality are organized into a cluster, where a more stable and powerful node is selected as header in each cluster and the others are at most 2 hops away from the header as the members of the cluster. All header nodes form a ring with Chord as group management protocol. An updating tree is built dynamically on top of the Chord ring to propagate the updated data items. In this way, the updated data item is broadcasted within cluster at MAC layer and transmitted among the header nodes along the updating tree. The simulation results demonstrate CCS outperforms the Gossip scheme for consistency of cooperative caching with less workload, higher success rate and less updating time.
Key words: MANET; cluster-based consistency scheme; cooperative caching; performance evaluation

摘 要: 协作缓存在移动自组织网络中得到了充分的应用和部署.提出了一种基于簇的一致性维护策略CCS(cluster-based consistency scheme).在CCS中,相邻的节点组成一个簇.每个簇中挑选一个能量较高、较稳定的节点作为簇头,而簇中的其他节点与簇头节点最多相距两跳.簇头节点利用基于DHT(distributed Hash table,分布式哈希表)的Chord协议作为组管理协议,即簇头节点组成一个Chord环.通过动态地在Chord环上建立更新树传播更新内容.这样,更新数据在不同的簇之间是通过更新树传播的,而在簇内是通过MAC层的广播传播的.仿真实验结果表明,与基于流言传播的缓存一致性维护策略相比,CCS具有开销小、成功率高和传播快的特点.
关键词: 移动自组织网络;基于簇的一致性维护策略;协同缓存;性能评估
中图法分类号: TP393 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant Nos.60403031, 90604015 (国家自然科学基金); the National Basic Research Program of China under Grant No.2007CB310702 (国家重点基础研究发展计划(973))

Received 2007-03-05; Accepted 2007-08-03

1 Introduction

Mobile Ad hoc networks (MANETs) are autonomously structured with multi-hop wireless links in peer-to-peer fashion without the aid of infrastructure network[1]. The rapid progress in wireless network, mobile communication and portable computer technologies makes MANETs to be used not only in military rescue scenarios, but also for industrial and commercial applications[2]. There are lots of previous researches in MANET focusing on MAC schemes and routing protocols[1]. Although MAC and routing are important issues in MANET, efficient data access is also very important since there are many restrictions, such as unpredictable signal propagation, limited bandwidth, dynamic topology and energy constraint, which introduce unique issues and difficulties for data delivery and application deployment in MANET environments[3]. Cooperative caching is a useful technique to improve the system performance to tackle these restrictions and implement cooperative computing[4,5].

Although cooperative caching has been extensively studied in Internet to improve the performance of applications, e.g., Web and P2P applications[6,7], these methods cannot be applied to MANETs directly due to routing, mobility and resource constraints[4]. Centralized scheme is a straightforward way to maintain replica consistency. However, if all nodes access data from the same source node, they update their cache at great expense of battery power, bandwidth, and traffic workload. A gossip-based scheme has a good quality of fault tolerance. However, it brings a lot of redundant update messages and consumes lots of bandwidth unnecessarily. Another feasible way is a tree-based scheme. This method has shorter convergence time and less redundant update messages. How to maintain the tree structure and improve the fault tolerance are the two most important problems in this scheme.

Cooperative caching and data replication schemes in ad hoc network have been studied in Ref.[4]. These approaches mainly focus on the generation of caches and replicas, but not on their consistency. In Ref.[5], the authors propose a relay peer-based caching consistency, to address the caching consistency issues in MANET. The data source chooses relay peers from stable, capable and powerful hosts, and push updates to these relay nodes periodically. Other caching hosts get the updated data from relay peers in a pull style. This hierarchical scheme augments the loads on the relay node, which would in turn overload these nodes. Further more, “pull” requests are broadcasted in the network, which consumes bandwidth and energy, the precious resource in MANET.

In this paper, we propose an efficient Cluster-based Consistency Scheme for cooperative caching, called CCS. CNs are grouped into different clusters. Any member nodes in a cluster can communicate with their header nodes at most 2 hops. Header nodes are organized in a Chord[8] ring. A binary tree, taking the information of location into account, is constructed on top of the Chord ring. Updated Data Items (UDIs) in members are submitted to their header nodes firstly and then transmitted along the updated tree, where the header node in the cluster with data source is the root node of the tree. Header nodes broadcast the UDIs to all the members in their clusters. Simulation results show that CCS generates less traffic workload and has higher updating success ratio. The updating tree is built dynamically in CCS so that it is unnecessary to maintain updating tree all the time. Although this may increase the updating time since construction of updating tree is required for every UDI, it is more cost-efficient than maintaining updating trees all the time when considering the dynamical nature in MANET.

The rest of this paper is organized as follows. Section 2 describes the service model of the MANET and CCS scheme. In Section 3, we analyze the performance theoretically. Section 4 simulates CCS and Gossip in ns2 and the experimental data are analyzed. Finally, we conclude and discuss our future work in Section 5.

2 CCS

2.1 System model

A MANET can be considered as a graph $G=\{V,E\}$, where $V=\{v_1,v_2,\ldots,v_N\}$ is the set of nodes in the network and $E=\{e_{i,j}|i,j=1,2,\ldots,N,i\neq j\}$ define the connection relationship between nodes. $e_{i,j}=1$ means v_j and v_i can send data each other directly, where we assume that if $e_{i,j}=1$ then $e_{j,i}=1$. There are several groups $\{G_1,G_2,\ldots,G_w\}$ in G, and all the nodes in the same group (say G_k) work cooperatively and keep caches consistent. Nodes belonging to the same group are called cooperative nodes (CNs). The number of nodes in group G_k is n_k. A node may belong to different groups since it can cooperate with others for different tasks. Once a node obtained an UDI, it should transmit it to its CNs with less traffic workload in less time.

Clustering scheme is proposed and deployed to improve the performance of routing protocol in MANET. It can decrease the scale of network and implement hierarchy routing[9]. A cluster C_i^k is a set of nodes close in geography location belonging to group G_k. So, $G_k=\{C_i^k \mid i=1,2,3,...,m_k\}$, where m_k is the number of clusters in the group. There is a header node $HN_0^{k,i}$ and several member nodes $MN_j^{k,i}$ in the cluster C_i^k. $MN_j^{k,i}$ transmits data to $HN_0^{k,i}$ firstly. Then, $HN_0^{k,i}$ transfers the data to other header nodes as well as the member nodes in the same cluster. The other header nodes will also broadcast the data to all of their members. A well-known example of cluster is the 3hBAC (3-hop between adjacent cluster heads)[10].

There are three procedures in CCS: Clustering Procedure in the initial phase, Updating Tree Constructing Procedure when an UDI emerges and Data Transmitting Procedure. We describe these procedures in this section.

2.2 Clustering procedure

Every group has the same procedure. Without loss of generality, we just discuss CCS in the group named G_k. Other groups can perform CCS in the same way independently. Before the procedure description, some messages are listed as follows.

1) *CST_REQ*: Message for requesting the membership of cluster, containing group ID in the payload.
2) *CST_GRT*: Message granting the node sending *CST_REQ* to take part in the cluster as a member or a guest, containing group ID and cluster ID (cluster header ID) in the payload. A guest is a node in the group with a hop from a member.
3) *CST_ACK*: Message for acknowledging the *CTS_GRT*.
4) *CST_DCL*: Message sent from a cluster header to its member nodes or from a member node to its guest periodically.
5) *CST_QUIT*: Message from a member node to its cluster header and backup header to notify the node will quit the cluster.
6) *CST_BH_REQ*: Message from a cluster header to invite its member to be the backup header of the cluster.
7) *CST_BH_ACK*: Message to the cluster header to express the willing of being the backup header.
8) *CST_BH_GRT*: Message from a cluster header to grant the node sending *CST_BH_ACK* to be its backup header.

Every node belonging to the same group G_k has six possible statuses $S=\{$*Init*,*Header*,*Backup_Header*,*Member*, *Guest*,*Down*$\}$ denoting that the node is Initial (not belong to any cluster), header, backup header, member, guest of a cluster, or invalid, respectively. In the initial phase, the status of every node is *init*. A new node being added into the MANET is also in the initial status. The node v_j in initial status broadcasts a message *CTS_REQ* to its

neighbor nodes in one hop to request the membership of a cluster and starts up a timer. If a header of cluster $HN_0^{k,i}$ receives this message, it sends a *CTS_GRT* message to accept v_j as its member. Once v_j receives *CTS_GRT* from $HN_0^{k,i}$ and decides to join the cluster of $HN_0^{k,i}$, it sends a message *CST_ACK* to $HN_0^{k,i}$ and sets itself into *member* status. Then, $HN_0^{k,i}$ records v_j as its member. If a member node $MN_x^{k,i}$ of a cluster C_i^k receives the message *CTS_REQ* from v_j, $MN_x^{k,i}$ also replies *CTS_GRT* message to v_j and asks v_j to be its guest. If v_j agrees to be the guest of $MN_x^{k,i}$, it replies *CST_ACK* to $MN_x^{k,i}$ and sets itself to a *guest* status. If the timer of v_j is time out and v_j has not received *CTS_GRT* from other nodes, v_j sets itself into *header* status and broadcasts *CST_DCL* in one hop. If a node receives the *CST_DCL*, it can choose to join the cluster and reply *CST_ACK*. The transform of nodes status is described in Fig.1.

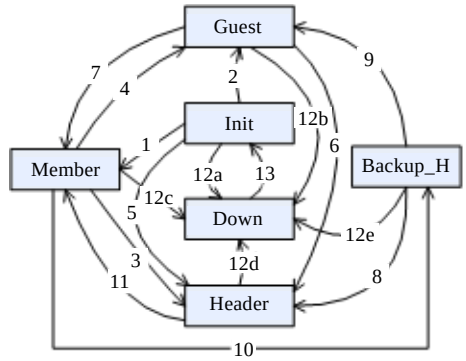


Fig.1 Node status transform

2.3 Updating tree of header nodes

In this section, we discuss the organization architecture of header nodes (HNs). In CCS, we take advantage of Chord[8] as the group management protocol for all HNs which form a Chord ring R_k logically. A new added HN, labeled as $HN_0^{k,i}$, joins the Chord ring according to the joining protocol of Chord[8]. When $HN_0^{k,i}$ is out of service or changes its header status, it leaves the Chord ring according to the leaving protocol of Chord[8]. If the UDI originated from the cluster of $HN_0^{k,i}$, a binary tree with $HN_0^{k,i}$ as its root is built on top of ring R_k to propagate the UDIs. In this section, we first illustrate how to build a Chord tree, which is locality irrelevant. Then, the method of adjusting a Chord tree to a corresponding locality aware updating tree is proposed.

2.3.1 Chord tree construction

As mentioned above, the online HNs form a Chord ring R_k. The Chord tree is built on top of R_k through partitioning the identifier space that the ring represents. Initially, the sponsor node $HN_0^{k,i}$ holds the whole identifier space. This identifier space is partitioned into two parts with equal size. We choose the first node of each part as the representative node to hold the identifier space of this part and set these two HNs as the children of $HN_0^{k,i}$. Each part is further partitioned into two parts with equal size, and so forth, until there is only one HN in this identifier part. The pseudo code is listed in detail in Fig.2. The notation *X.foo*() stands for the function *foo*() being invoked at and executed on HN *X*. The function *find_successor*(*id*), provided by the Chord protocol, is used to find the successor node with the *id*. The function *X.get_rpn*(*region*) is to get the representative node of the *region* space.

```
X.region_partition(region_x)
1:  if (X.id+1>region_x.end)
2:      return;
3:  region←(X.id+1,region_x.end);
4:  Split region into 2 partitions with equal size
5:  for i=1 to 2{
6:      region[i]←the i-th partition;
7:      RPN_region[i]=X.get_rpn(region[i]);
8:      if (RPN_region[i]!=NULL){
9:          X.children=X.children∪RPN_region[i];
10:
RPN_region[i].region_partition(region[i]);}
11: } /* end of for i=1,…*/

X.get_rpn(region)
1:  id←first ID of this region;
2:  node←X.find_successor(id);
3:  if (node.id∉region)
4:      return NULL;
5:  return node;
```

Fig.2 Chord ring building

Suppose there are HNs list in a MANET denoted as $\{v_0,v_1,v_2,v_3,v_4,v_6,v_7,v_{10},v_{12},v_{13}\}$. Figure 3 illustrates a Chord ring and a binary tree built on top of this ring according to the algorithm presented in Fig.2.

Lemma 1[11]. For a Chord ring with m_k HNs, the maximum

number of hops from the root node to the leaf node for UDI transmission along the Chord tree in the MANET, with high probability, is

$$H_c' = O\left(\log_2 m_k \times \frac{\sqrt{a}}{r}\right) \tag{1}$$

where a is the acreage of the MANET, r is the coverage radius of a node.

In CCS, the maximum hops of UDI transmission in a cluster are 2 hops, from guest nodes to HN or reverse direction. It is straightforward to deduce Theorem 1 below.

Theorem 1. The maximum number of hops for UDIs with CCS with the highest probability is less than

$$H_t' = O\left(\log_2 m_k \times \frac{\sqrt{a}}{r}\right) + 4 \tag{2}$$

If an UDI in a cluster is transmitted to an HN, the UDI can be transmitted along the Chord tree where the initiative HN is the root node. However, the Chord ring is locality irrelevant as it does not account for the proximity relationship among header nodes in the consistent hash function. Thus, the Chord tree, which is built on top of the Chord ring, is also locality irrelevant. So, there is a topology mismatch between Chord tree and actual MANET topology. Propagating UDIs with this Chord tree costs unnecessary bandwidth and energy. For example, in the Chord tree illustrated in Fig.3, it is possible that v_2 and v_6 are neighbors of v_0 in MANET, and v_3 and v_7 are neighbors of v_1 in MANET. In this situation, an UDI may follow the path $\{v_0,v_6,v_1,v_6,v_7\}$ from the network layer view in MANET. Then, v_6 needs to receive and forward the UDI twice, the first one is forwarded to v_1, and the second one is for itself. But only the last one makes sense for data updating of v_6 in the application level. Note that when v_1 sends data to v_6, v_7 may also receive a copy of the data for that it is a neighbor of v_1. However, only the network layer of v_7 can perceive the data and the application layer has no knowledge of the data. This motivates us to exploit the locality information of MANET and adjust the Chord tree to a more efficient updating tree.

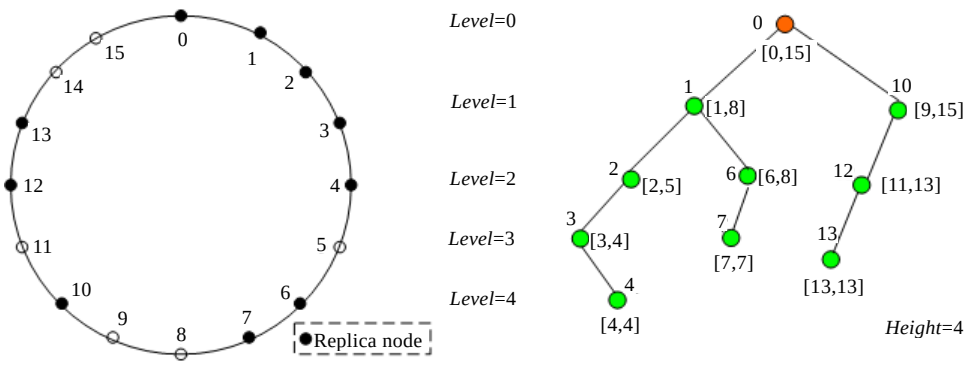


Fig.3 Chord ring and the chord tree

2.3.2 Updating tree construction and UDIs transmission

As in Ref.[11], we define some types of messages for the procedures of updating tree construction and UDIs transmission.

- *UTC_Init_Msg*: Initializing message of updating tree construction which is generated by the root HN.
- *BDReq_Msg*: Binding requirement message is sent from a node requiring another node to be its parent node in updating tree.
- *RLSReq_Msg*: Releasing requirement message is sent from a child node to its parent node in Chord tree to end their joint relationship and ask its parent node to take over the children nodes.

A table is stored in each HN with a list of dynamic data items consisting of a group ID, a root node ID, a parent node ID, a grandparent ID and a list of children nodes. Initially, the table denotes the topology of a Chord tree. After the procedure of updating tree construction, the table denotes the topology of the updating tree.

A cross-layer method is proposed to adjust a Chord tree to an updating tree in order to reduce the topology mismatching. The root node in Chord tree initially sends an *UTC_Init_Msg* to all its children nodes once a UDI emerges on it. Its child HN v_m receives, parses and validates the *UTC_Init_Msg*. If it is a valid message, v_m checks whether the value of field of *HN_ID* v_p in *UTC_Init_Msg* is the ID of its parent node v_q of Chord tree. If v_p=v_q, v_m sends a *BDReq_Msg* to v_p as its parent node in the updating tree, then forwards the message to its children HNs. Once v_p receives the *BDReq_Msg*, it adds v_m as one of its child nodes and replies an acknowledgement message to v_m. Otherwise, besides *BDReq_Msg* from v_m to v_p, v_m sends an *RLSReq_Msg* to v_q, and asks for v_q to delete its ID from child nodes list of v_q and take over its child nodes. v_q adds child nodes of v_m into *Children_ID* of its table and sends message to these children nodes of v_m to update the fields of *Parent_ID* and *GParent_ID* with v_q and parent node ID of v_q in their tables. In the aforementioned example, after adjustment, node v_6 will be a child of v_0. Thus, the *UTC_Init_Msg* sent by v_0 to v_7 follows the path $\{v_0,v_6,v_1,v_7\}$ from the network layer. Note that, from the application layer, both v_1 and v_6 are children of v_0. If we suppose that v_6 is an intermediate node in the path between v_0 and v_7, v_4 is an intermediate node in the path between v_{10} and v_{12}, and v_{13} is an intermediate node in the path between v_2 and v_3, then the corresponding updating tree of the Chord tree of Fig.3 is shown in Fig.4 (the paths from 0 to 1, 0 to 10, 1 to 2, 1 to 7, 2 to 3 and 10 to 12 are {0,6,1}, {0,10}, {1,2}, {1,7}, {2,13,3} and {10,4,12}, respectively). The procedure of a updating tree constructing can be found in Ref.[11].

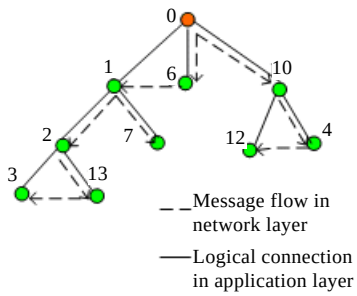


Fig.4 Updating tree

When we build updating tree from Chord tree, while the HNs in Chord tree may not be promoted to an upper level, at least they are not demoted to a lower level. Thus, the height of updating tree is less than the height of the Chord tree. With respect to the average height of updating trees, we have the following theorem.

Theorem 2. The upper bound of average height of the updating trees is not longer than Chord tree.

$$H_t \le O\left(\log_2 m_k \times \frac{\sqrt{a}}{r}\right) + 4 \tag{3}$$

Compared with the Chord tree, although the height of the updating tree may not be reduced, the number of hops from the network layer view can be reduced greatly as the HNs which are close in the updating tree are close in the actual MANET too.

Once the updating tree has been constructed, the UDI is submitted to the HN in the same cluster firstly and then propagated from the HN, along the updating tree, to all the other HNs. Finally, the other HNs transmit the UDI to their member nodes. The clusters are built and maintained all the time, but the updating tree is constructed once an UDI appears and destroyed after the process of updating. The updating tree is released in the following way. Once the root HN finishes the transmission the UDI to its child HNs, it deletes the information of the updating tree from its memory.

Note that after transferring the update data, the links between the parent HNs to their children nodes are destroyed. The reason is that, to fully take advantage of the system resources, we should use multiple updating trees to propagate updated contents. Otherwise, if we use only one updating tree, most nodes are leaf nodes which would make no contribution to content delivering. In our experiments, the maintenance cost of multiple updating trees is

larger than the cost of building an updating tree while necessary, especially in dynamic MANET systems

2.4 Other issues in CCS

2.4.1 Roaming of nodes

In MANET, nodes can roam from one area to another. The situation of nodes out of service due to nodes roam is described later. If the mobile node is a member or a guest, it can be maintained as clustering procedure. If the node is an HN, although the HN roam may change the route path between HNs, it has no effect on the Chord ring because the HNs are still reachable from other HNs. However, the updating tree may not be as efficient as before. Note that, although this brings some unnecessary overhead, it does not disrupt the update operation. And in view of the fast convergence of CCS, we do not address this issue, leaving it for future work.

2.4.2 Invalidation of nodes

If the invalid node is a member node or a guest node, it can be processed according to the clustering procedure and doesn't affect other clusters. The MN sends a leave request to its header node and the HN deletes the MN from its member list. If the HN detects that one of its MN is invalid, it also deletes the MN in its memory.

If the invalid node is an HN, the situation is more complicated. Every HN stores q prevenient HNs of the updating tree in its memory. A periodic detection message is sent from HNs to their parent HN. If the HN finds its parent HN is invalid, it sends a *BDReq_Msg* to its grandparent HN and sets its grandparent to its parent node. If it does not receive acknowledgement from all its prevenient HNs by some deadline, it sends a message to a root node for rebuilding a Chord tree. Then, the Chord tree will be reconstructed.

2.4.3 New nodes

New nodes can join current MANET dynamically. If a new node is interested in the UDI and wants to maintain consistency with other nodes, it tries to find a cluster, joins it and gets UDI from the HN. Otherwise it joins as an HN into the Chord ring according to the Chord joining protocol. Then, it asks the mobile nodes stored in its finger table[8] for the latest version of the UDI. This is similar to the "pull" scheme in Ref.[6].

3 Theoretical Assessment of CCS

In this section, we analyze the performance of CCS furthermore and define some performance metrics of the consistency scheme for the simulation experiments.

3.1 Performance of CCS

Every HN in G_k can generate UDIs and initiate construction of an updating tree as the root node. Several UDIs at the same node can be transmitted along the same updating tree. So, only one updating tree is necessary for several UDIs at the same node. That is, the maximal number of updating tree concurrently existing in group G_k is the same as the number of clusters. So, we can get some similar conclusions to that in Ref.[11].

Theorem 3[11]. The maximal number of data items in the table storing the updating tree in each HN is

$$DI=m_k \tag{4}$$

Theorem 4[11]. The total number of hops of a UDI transmission on average is

$$H'_u=\frac{(m_k-1)\times\sqrt{\frac{a}{\pi\times m_k}}}{r}+2 \tag{5}$$

3.2 Performance metrics

The goal of the consistency scheme is keeping CNs consistent with the least traffic and least time even in unstable and dynamic network. In order to verify the performance of the consistency scheme, some metrics are

defined below.

- The overhead of updating is defined as the overhead the scheme brings to the network. The overhead of updating *OU* is defined as Eq.(6)

$$OU = \sum h_i \times B_i \tag{6}$$

where B_i is bytes of message *i*, and h_i the number of hops message *i* traversed.

- We define the workload of updating, including traffic in control plane and data plane as Eq.(7)

$$WU = OU + DU = \sum h_i \times B_i + \sum h_j \times B'_j \tag{7}$$

where B'_j is the packets of UDI_j and h_j is the total hops for UDI_j.

- In order to estimate the updating speed, we define the period from UDI generation to updating finished as the time of updating.
- Due to mobility, invalidation or island nodes, it is unavoidable that some HNs fail to be updated with UDI. The success rate of updating is defined as Eq.(8)

$$R = \frac{n'_i}{n_i} \tag{8}$$

where n'_i is the number of nodes updated successfully.

4 Simulation Experiment

The performance of CCS has been evaluated through simulation experiments under different system settings. We also compare CCS with gossip scheme[12] and our previously proposed DTCS scheme[11]. In gossip scheme, When a CN in a group receives a UDI for the first time, it chooses *k* CNs from the group randomly, and sends the UDIs to these CNs. Otherwise, it just simply discards the UDIs that have been received before. To achieve a fair comparison, the fanout of each node in Gossip is limited to two (i.e. *k*=2). DTCS distinguishes from CCS that it has no cluster and all the CNs are formed in a Chord ring.

We mainly focus on three metrics: updating workload, updating time and success rate, defined in Section 3.

4.1 The simulation model

The simulation is based on ns-2[13] in version 2.28 with the CMU wireless extension running on Dawning 4000A super computer with 44 64-bits Opteron CPU of AMD, 84GB memory and Suse Linux Enterprise Server 9.0 operating system with 2.6.5-7.97-smp kernel. AoDV[14] is used as the underlying routing protocol. The default simulation parameters are shown in Table 1. We assume that only some nodes are involved in the cooperative work (e.g. only these nodes have caches of data) and the cooperative nodes are chosen randomly from the system. The node moving pattern follows the random way point movement model[4]. The UDI is generated randomly in CNs. Finally, the HNs broadcast CST_DCL message every 5 seconds.

Three experiments are performed with the same experiment parameters but different scenario scripts. The average values of the three experiments are calculated and reported.

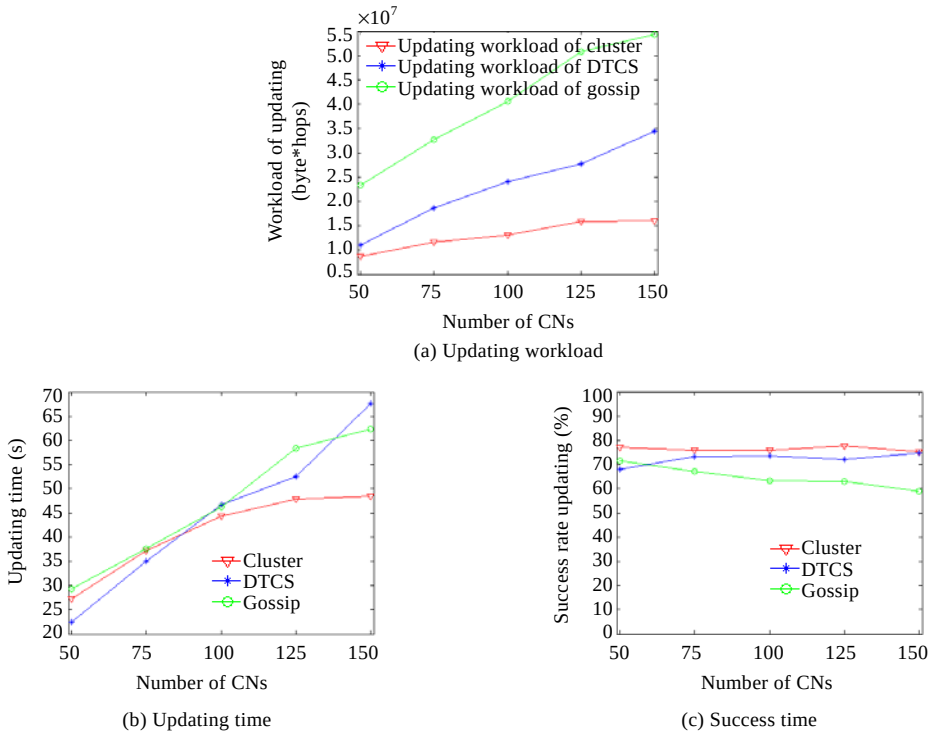
Table 1 Simulation parameters

Parameter	Default value	Range
Number of nodes	300	200~400
Number of CNs	100	50~150
Area (km×km)	2×2	1×1~3×3
Propagation distance (m)	200	100~300
v_{max} (m/s)	2	2~18
Pause time (s)	30	–
Simulation time (s)	600	

4.2 Effects of number of CNs

In Fig.5, the performance of each scheme with different number of CNs has been estimated and compared. It is apparent that the updating workload and updating time of each scheme increase linearly as the number of CNs grows. The success rate of each scheme seems to have little relationship with the number of CNs. Compared with Gossip and DTCS, CCS has achieved a higher success rate and lower updating time, but with less updating workload. There are two reasons for that, 1) In CCS, only the headers join the chord tree to get their replica, which limited the height of the chord tree and made the scheme converge rapidly; 2) the members get their replicas by the headers' broadcast, which also save a lot of bandwidth.

Note that when the number of CNs is small, the updating time required in CCS is bigger than in DTCS. This can be explained by two facts. First, when the number of CNs is small, the success rate of CCS is bigger than the success rate of DTCS. This shows that CCS needs to send data to more CNs than DTCS, which increases the updating time. Second, in CCS, the source node only sends the updating data to one node (i.e. its cluster header) at the first step, while, in CCS, the source node can send the updating data to several nodes. And when the number of CNs is small, this first step counts more.



(a) Updating workload

(b) Updating time

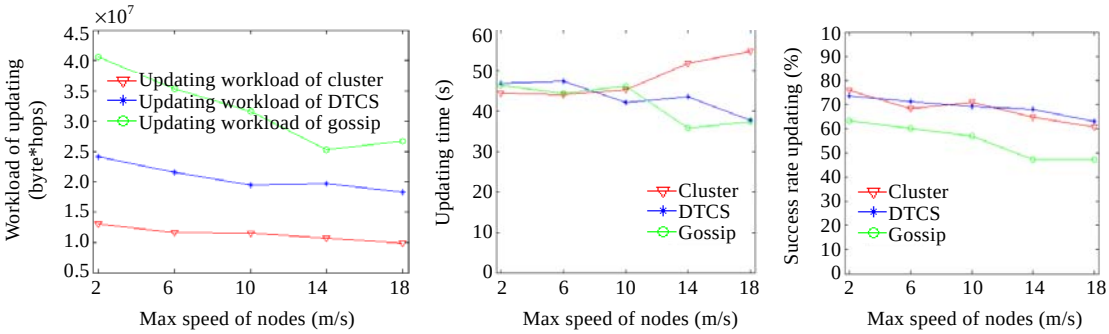
(c) Success time

Fig.5 Effects of number of CNs

4.3 Effects of moving rate of nodes

The movement of nodes may also affect the performance of a scheme. Figure 6 compared each scheme under different movement of nodes. It is obvious that the success rate starts to decrease as the nodes move frequently, which also causes decreasing of updating workload. The updating time seems to have nothing to do with the movement of nodes. To achieve the equal performance, CCS needs less updating workload, compared with two other counterparts. Note that the success rate decreases slightly with the increasing of moving rate. Thus, the

number of nodes that receive the update data significantly decreases, which, in turn, causes the decreasing of updating workload as shown in Fig.6(a).



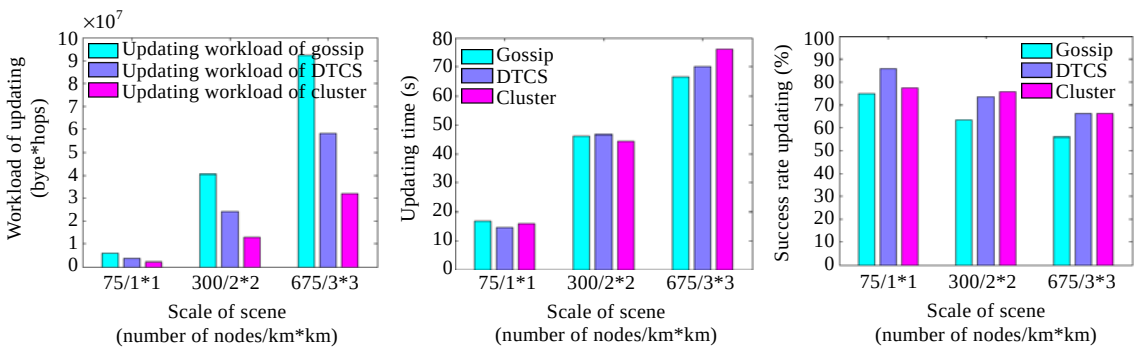
(a) Updating workload (b) Updating time (c) Success time

Fig.6 Effects of moving rate of nodes

4.4 Effects of network scale

Here the Scale of scene means that only the size of scene varies from 1km*1km to 3km×3km, at the sametime denisity of nodes and CNs. Thus, as the scale increases, the number of CNs also increases. Figure 7 gives the intuition that how the scale of scene affects the performance of a scheme. As the scale of scene grows, the updating workload and the updating time increase linearly. At the same time the success rate drops linearly. In each scale of scene, CCS seems to need the least workload to achieve the same performance as that in the other two schemes.

Note that the Success Rate decreases slightly with the increasing of network scale. This can be explained as follows. First, because the density of nodes and CNs remain unchanged, the number of CNs increases. This inevitably increases the length of the updating path. As the path length increases, the probability of failure increases.



(a) Updating workload (b) Updating time (c) Success time

Fig.7 Effects of network scale

5 Conclusions

A consistency scheme of cooperative caching, called CCS, is proposed for MANET. In this scheme, all group members are organized into several clusters. An updating tree is constructed dynamically for cluster headers with Chord. UDIs can be transmitted along the updating tree and broadcasted in the clusters. The overhead of maintaining the ring is slight since it is not necessary to justify the Chord ring with the dynamic topology due to the roaming of nodes. The updated tree is computed from the ring referring to the routing information and released after

the UDI transmission. The scheme can meet the characteristics of topology dynamic, mobility of node and energy limitation. The results of simulation experiments with different settings show that the scheme can keep caches consistent with less workload and higher success rate of updating.

References:

[1] Corson S, Macker J. Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations. RFC 2501, 1999.

[2] Huang E, Hu WJ, Crowcroft J, Wassell I. Towards commercial mobile ad hoc network applications: A radio dispatch system. In: Proc. of the 6th ACM Int'l Symp. on Mobile Ad Hoc Networking and Computing. Urbana-Champaign Illinois, 2005. 355−365.

[3] Mohapatra P, Li J, Gui C. QoS in mobile ad hoc networks. IEEE Wireless Communications Magazine, 2003,10(3):44−52.

[4] Yin LZ, Cao GH. Supporting cooperative caching in ad hoc network. IEEE Trans. on Mobile Computing, 2006,5(1):77−89.

[5] Cao JN, Zhang Y, Xie L, Cao GH. Consistency of cooperative caching in mobile peer-to-peer systems over MANET. In: Proc. of the 25th IEEE Int'l Conf. on Distributed Computing Systems Workshops. 2005. 573−579.

[6] Datta A, Hauswirth M, Aberer K. Updates in highly unreliable, replicated peer-to-peer systems. In: Proc. of the IEEE ICDCS 2003. 2003. 76−85.

[7] Kahol A, Khurana S, Gupta SKS, Srimani PK. A strategy to manage cache consistency in a disconnected distributed mobile wireless environment. IEEE Trans. on Parallel and Distributed Systems, 2001,12(7):686−700.

[8] Stoica I, Morris R, Karger D, Kaashoek M, Balakrishnan H. Chord: A scalable peer-to-peer lookup service for Internet applications. In: Proc. of the SIGCOMM 2001. 2001. 149−160.

[9] Yu JY, Chong PHJ. A survey of clustering schemes for mobile ad hoc networks. IEEE Communications Surveys and Tutorials, 2005,7(1):32−48.

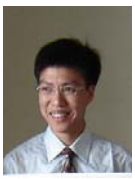
[10] Yu JY, Chong PHJ. 3hBAC (3-hop between Adjacent clusterheads): A novel non-overlapping clustering algorithm for mobile ad hoc networks. In: Proc. of the IEEE Pacrim 2003. 2003. 318−321.

[11] Xie GG, Li ZY, Chen JN, Wei YF, Issarny V, Conte A. DTCS: A dynamic tree-based consistency scheme of cooperative caching in mobile ad hoc networks. In: Proc. of the IEEE WiMob 2007. 2007.

[12] Ganesh AJ, Kermarrec AM, Massoulie L. Peer-to-Peer membership management for gossip-based protocols. IEEE Trans. on Computers, 2003,52(2):139−149.

[13] Homepage of ns2. 2007. http://nsnam.isi.edu/nsnam/index.php/Main_Page

[14] Perkins C, Belding-Royer E, Das S. Ad hoc on-demand distance vector (AODV) routing. RFC 3561, 2003.



XIE Gao-Gang was born in 1974. He is a professor at the Institute of Computing Technology, Chinese Academy of Sciences and a CCF senior member. His research areas are distributed and mobile computing, network test and measurement.

LI Zhen-Yu was born in 1980. He is currently working toward the Ph.D. degree at the Institute of Computing Technology, Chinese Academy of Sciences. His research areas are mobile computing and P2P.

CHEN Jia-Ning was born in 1982. He is currently working toward the MS degree at the Hunan University and doing visiting research in ICT. His research areas are mobile computing and network simulation.