Consistency of Cooperative Caching in Mobile Peer-to-Peer Systems over MANET

Jiannong Cao[1], Yang Zhang[1, 2], Li Xie[2] and Guohong Cao[3]
[1]Dept. of Computing, Hong Kong Polytechnic University, Hong Kong
[2]Dept. of Computer Science & Technology, Nanjing University, P.R.China
[3]Dept. of Computer Science & Engineering, The Pennsylvania State University, USA
Email address for correspondence: csjcao@comp.polyu.edu.cn

Abstract

Cooperative caching can be used to improve the availability and scalability in data sharing and dissemination as well as to reduce the network traffic and query delay. Caching consistency is an important issue in cooperative caching but has not been adequately addressed for mobile ad hoc networks (MANETs). In this paper, we proposed a novel approach, called RPCC (Relay Peer-based Cache Consistency), to addressing the caching consistency issues in a MANET. With the introduction of relay peers between the source host and the cache nodes, both push-based and pull-based strategies can be employed, which helps to reduce the communication overhead and query latency. The source host pushes the data to the relay peers and the cache nodes pull the data from the relay peers. These operations can be performed asynchronously and simultaneously. Moreover, the proposed RPCC approach is flexible in that it can deal with three kinds of consistency requirements (strong, Δ and weak consistency) adaptively. Simulation results demonstrated that RPCC outperforms the traditional push and pull based strategies by taking their advantages while at the same time avoiding their weakness.

1. Introduction

A MANET is a network without a fixed network infrastructure but just consisting of a collection of autonomous mobile hosts that communicate over relatively bandwidth constrained wireless links [Cor99]. Similarly, in a peer-to-peer system, no node acts explicitly as a central server and all the peers need to collaborate with each other in order to make the whole system work [Din 04]. Because of the same decentralized property, it is natural to develop and deploy applications over a MANET using the peer-to-peer model. Various functions such as resource sharing and collaboration can be achieved through direct interactions between peers. In this paper, we call a peer-to-peer system over a MANET a *Mobile Peer-to-Peer System (MP2P)*.

Data dissemination and sharing among peer nodes in a MP2P network is an important problem and has many applications. The following are just some examples:

- In a battlefield, a group of soldiers, each with a micro-data center and related communication tools, can form a mobile ad hoc network. The soldiers update the information (e.g. geographic information or enemy information) in their data centers momentarily, and can share with each other the new information and commands.
- A mobile store system consists of several mobile booths that store the information (e.g. price, sum, etc) of the commodities. People can visit any mobile booth to select the commodity they want. The booths having the data item cache of the same commodity will need to exchange the deal information with each other.
- A MP2P network can be deployed to provide mobile users access to Internet services. Due to limited radio range, a wireless access point on the Internet cal only cover a limited geographical area, so some mobile users moving out of the coverage range will not be able to access the Internet. However, if mobile users can form an ad hoc network, they can still gain the access via other peer nodes within the coverage range.

However, developing high performance techniques for data transmission in a peer-to-peer system over a MANET is a difficult task [Cao04]. This is mainly due to the following salient characteristics of a MP2P system.

•*Dynamic Topology*: Each mobile host can roam freely in the network and leave/rejoin network at any time and any where. As a result, the network topology may change randomly and dynamically.

•*Peer-to-peer Type*: Unlike the traditional Client/Server mode, there is no centralized control in the network and all the mobile hosts in the network are symmetric, communicating directly with each other in multi-hop fashion.

•*Energy Constraint*: The battery power of a mobile host is limited. Energy should be reserved whenever possible in order for the mobile host to work longer.

•*Limited Bandwidth*: Wireless links always have lower capacity than the wired links. Moreover, the wireless channel is less stable with greater variants in congestion and higher packets loss rate.

To tackle above problems, a useful technique called *Cooperative Caching* can be used to improve the system performance [Cao04, Cho04]. Cooperative caching allows the sharing and coordination of cached data among multiple nodes in the network. By cooperatively caching frequently accessed data items, mobile users can access to the data items by communicating among themselves without always having to send requests to the data source. This leads to a shorter user response time and query latency, and less communication overhead and energy consumption of mobile hosts.

Cooperative caching needs to address three major issues: *placement*, which determines where to place copies of data items and when to replace them; *discovery*, which directs the query request to an appropriate peer that copies the requested data item; and *consistency*, which maintains the desired level of consistency among the various copies of a data item. Most researches are concerned with the former two issues. In this paper, we focus on the cache consistency issue.

Maintaining cache consistency means to ensure that each caching node of a data item be aware of the data update when the source data changes. Cache invalidation techniques have been widely studied in the context of traditional wired distributed systems [Nel 88, How 88, Gra89, Gwe96, Liu97, Yin99] and infrastructure-based mobile wireless networks [Bar94, Wu96, Jin97, Kah01, Cao03, Wan04]. However, these Client/Server-based techniques are not directly applicable in MP2P systems due to the above-mentioned special properties of a MP2P network. In an infrastructure-based wireless network environment, caching consistency can be achieved through one-hop Invalidation Report (IR) from Mobile Support Station (MSS). However, in a MP2P system, the MSS may be unreachable in one single hop thus is undependable for caching invalidation. Meanwhile, all source data are dispersed across the MP2P network, which makes the cache invalidation much more complex. Therefore, new cache invalidation techniques are needed for MP2P systems.

In this paper, we propose a novel cache invalidation approach, called *relay peer-based strategy (RPCC)*, for maintaining consistency of cooperative caching in MP2P networks. With the introduction of relay peers between the source node and the cache nodes, both push-based and pull-based strategies can be employed, which helps to reduce the communication overhead and query latency. The source node pushes the data to the relay peers and the cache nodes pull the data from the relay peers. These operations can be performed asynchronously and simultaneously. Moreover, the proposed RPCC approach is flexible in that it can deal with three kinds of consistency requirements (strong, Δ and weak consistency) adaptively. The results of our performance evaluation study demonstrate that RPCC can greatly outperform the conventional simple push and pull based strategies by taking their advantages while at the same time avoiding their weakness.

The rest of this paper is organized as follows. In Section 2, we briefly overview the existing work on cache consistency. Section 3 describes the system model and provides the background of this paper. The proposed RPCC approach is presented in Section 4. Section 5 reports the results of performance evaluation of RPCC. Finally, we conclude the paper in Section 6 with a discussion of our future work.

2. Related Work

Caching invalidation techniques have been widely used in distributed systems for maintaining data consistency among caches [Nel88, How88, Gra89, Gwe96, Liu97, Yin99]. However, these classical techniques are not suitable for a mobile environments due to frequent disconnections and high mobility of mobile clients [Cao02].

There are two types of mobile wireless networks: *Infrastructure-based* and *ad hoc-based*. An infrastructure-based wireless network uses fixed network access points (e.g. Mobile Support Station, MSS) to forward messages that are sent/received by mobile hosts. As Figure 1 illustrates, the MSS is more like a server in the traditional Client/Server distributed system in that all the source data are deployed on it. Other mobile hosts retrieve data from the MSS and may cache a replica by themselves. Consistency maintenance mechanisms for this model have been studied extensively, most of them are based on broadcasting by the MSS. Barbara, and Imielinksi [Bar94] proposed three strategies: *Timestamps (TS)*, *Amnesic Terminals (AT)*, and *Signature (SIG)*. In these strategies, the MSS broadcasts an *Invalidation Report (IR),* which indicates the updated data items. Rather than querying the MSS directly regarding the validation of the cached copies, the mobile host can listen to these IRs over the wireless channel. But due to the limitation of the size, an IR can only record the update information in *k* IR intervals, thus is useless to any mobile host that has been disconnected longer than *k* IR intervals.

To handle the long disconnection problem, Wu et al. [Wu96] and Jing et al. [Jin97] proposed their new schemes that can tolerate a long disconnection. In these IR-based schemes, the querying mobile host has to listen

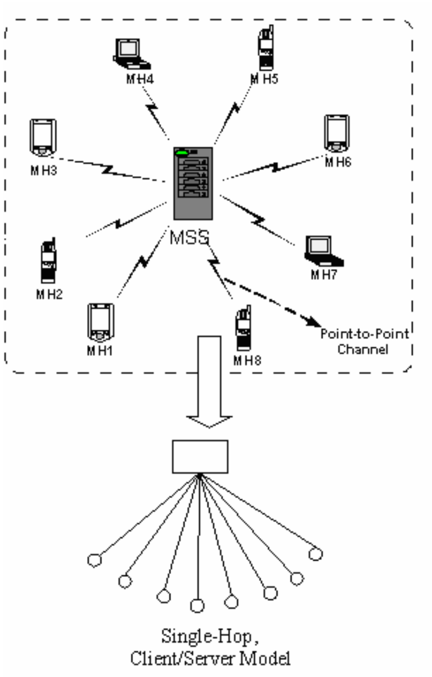


Fig 1. Infrastructure-based Mobile Network

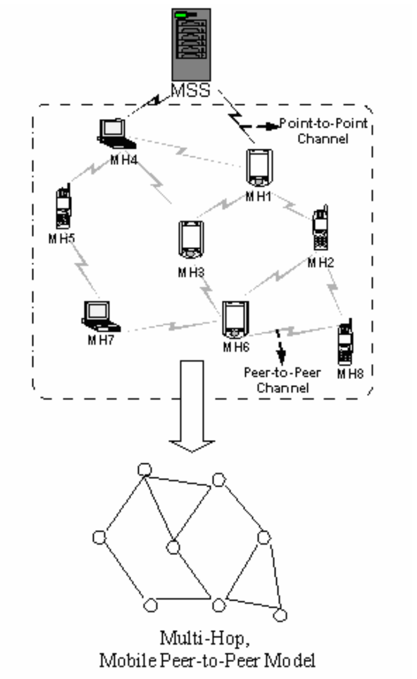


Fig 2. Ad Hoc based Mobile Network

to the next IR to conclude whether its cache is valid or not.
So the average waiting time is half of the IR interval. Cao [Cao03] designed an invalidation strategy that can reduce the query latency by inserting several updated invalidation report (UIR) between two successive IRs. Kahol, et al. [Kah01] proposed an asynchronous stateful (AS) strategy to maintain cache consistency. In AS, the MSS only broadcasts the data update to the related cache hosts and can avoid unnecessary IRs. Thus each mobile host can check its cache consistency asynchronously and the long disconnection problem is solved. But in this strategy, the MSS must record the state of all caches which cost too many resources of MSS. Wang et al. [Wan 04] proposed a strategy called Scalable Asynchronous Cache Consistency Scheme (SACCS), which improves the AS strategy. In SACCS, MSS only keeps minimum state information instead of all data and hosts information in AS. Therefore, the scalability and the performance can be greatly improved but stale data may exist in the caches.

However, cache invalidation strategies employed in a one-hop wireless mobile network are not suitable for MP2P systems over a mobile ad hoc network. In an MP2P system, even there is a MSS connecting the ad hoc network to the Internet (see Figure 2), the MSS may not be reachable in one single hop. Moreover, the MSS is used only as the access point to the Internet and will not keep data on it. All data items for searching and querying are dispersed across the mobile hosts in the ad hoc network. Each mobile host may have both the source data and cache copies of some data items from other hosts. In this model, the distance between the source data and its cache copy may be several hops and all the mobile hosts have to work cooperatively to keep the cached data item consistent with its source data.

Researches on cooperative caching for the ad hoc network model are mainly focused on caching replica allocation [Har02b, Har01, Cho04, Zhe04] or caching content selection [Cao04]. Work on cache consistency issue is quite limited. Lan et al introduced the consistency maintenance technique used in a Gnutella-like peer-to-peer file sharing networks [Lan03]. They proposed three strategies to keep the cache consistency: *simple push*, *simple pull* and *push with adaptive pull*. Lim et al also proposed a cache invalidation mechanism named GPSCE in MANET [Lim04]. GPSCE can improve the cache invalidation performance with the help of a global positioning system (GPS), but the expensiveness of GPS devices restricts its popularization. Our goal is similar to theirs but the RPCC approach proposed in this paper is more flexible, efficient and easier to implement.

3. System Model

A MP2P system consists of a collection of mobile peer nodes interconnected through a MANET. There is no central server to maintain cache consistency. The only central server named MSS is simply used to connect the MP2P network with outside wired network like the Internet. In this paper, the terms "host", "node", and "peer" are used interchangeably. Each mobile host can roam in the system freely and communicates with other mobile hosts through a wireless link using infrared transmission, radio frequency (RF) signals or via IEEE802.11/Bluetooth.

Each mobile host is assigned a unique identifier in the system. The set of all mobile hosts in the system is denoted by $M=\{M_1, M_2, \dots M_m\}$, where m is the total number of mobile hosts and M_i $(1 \le i \le m)$ is the identifier of host i. Each data item has a unique identifier. The set of all data items is denoted by $D=\{D_1, D_2, \dots D_n\}$, where n is the total number of data items and D_i $(1 \le i \le n)$ is the identifier data item i. Each data item also has a unique source host, called the source host, and the copy of the data stored at the source host is called the *master copy*. For simplicity and without loss of generality, we assume that the source host of data item D_i is M_i, thus $m=n$ and each mobile host holds one master copy of some data item, serving as its source host. A data item may have several replicas, called a *cache copies*, stored at its cache nodes. Each mobile host can serve as the cache nodes for *C-Num* data items. Modifications to a data item can be made only by its source host. In other words, only the *master copy* can be modified and the source host always has the most up-to-date version of the data item. The version number is set to zero when the data item is created and is incremented on each subsequent update.

Three levels of caching consistency can be defined: *strong-consistency (SC)*, Δ*-consistency (DC)*, and *weak-consistency (WC)*. Let $S^t_{D_i}$ denotes the version number of data item D_i at the source host and $C^t_{D_i,M_j}$ denotes the timestamp value at time t of data item D_i at the cache node M_j. We can formally define the three consistency levels as follows.

Strong-consistency: The version of data item D_i at any cache node is always up-to-date with the source data at the source host when a query request is served. That is, we have

$$\forall t,\ \forall j,\ C^t_{D_i,M_j} = S^t_{D_i} \qquad (3.2.1)$$

Δ*-consistency:* Any read of the cached copy of a data item is never out-of-date by more than Δ time with the master copy at the source host. That is, the stale time is within a bounded distance (i.e., Δ). We have

$$\forall t, \forall j, \exists \tau, 0 \le \tau \le \Delta, s.t.\ C^t_{D_i,M_j} = S^{t-\tau}_{D_i} \qquad (3.2.2)$$

Weak-consistency: A read of the cached copy of a data item at cache node D_i will return some previous correct value but does not necessarily reflect the most up-to-date version of the source data at the source host. That is, we have

$$\forall t, \forall j, \exists \tau,\ s.t.\ C^t_{D_i,M_j} = S^{t-\tau}_{D_i} \qquad (3.2.3)$$

Finally, we assume that the system has an independent mechanism for replica placement and for locating the nearest cache node to access the data copy.

4. A Relay Peer Based Approach to Caching Consistency

Both the source host and cache nodes can initiate consistency invalidation. The source-initiated approach is *push* based. The source host pushes the invalidation messages to the cache nodes. On the contrary, the cache peers-initiated approach is a *pull* based. A cache node polls the source host to determine whether its cached copy is stale or not. Push-based cache invalidation strategies are more suitable to a stable network, and can provide good consistency guarantees for hosts that are online and reachable from the source host. But they may suffer a long query latency and cannot solve the disconnection problem. If a cache node is disconnected from the network, it cannot receive the invalidation messages and will contain the stale data upon reconnection. On the other hand, pull-based strategies are suitable for dynamic networks but it brings large communication overhead. Meanwhile, the on-demand polling by cache nodes will consume more battery power. In this section, we present a the RPCC approach that can combine the advantages of both push-based and pull-based strategies while avoiding their weakness so as to provide a high performance for caching consistency maintenance in a MP2P system.

4.1. Overview of the RPCC Approach

Comparing with the conventional caching approaches, in RPCC, a new overlay consisting of some nodes with high capabilities, called relay peers, is created between the source host and the cache nodes. The source host and the relay peers can form a stable overlay network. The source host sends invalidation messages to its relay peers periodically. The communication between the source host and the relay peers are push-based since the overlay network is relatively stable. At the same time, because the network connections between the relay peers and other cache nodes are more dynamic, the communication between cache nodes and relay peers is pull-based. Thus, if a cache node has been disconnected from the network, upon reconnection, it only needs to find the nearest relay peer to check the status of the data items cached by it. Because the *push* and *pull* operations can be performed simultaneously and asynchronously, the query latency and communication overhead can be reduced. Moreover, with the use of relay peers, query requests with different consistency requirements can be handled adaptively.

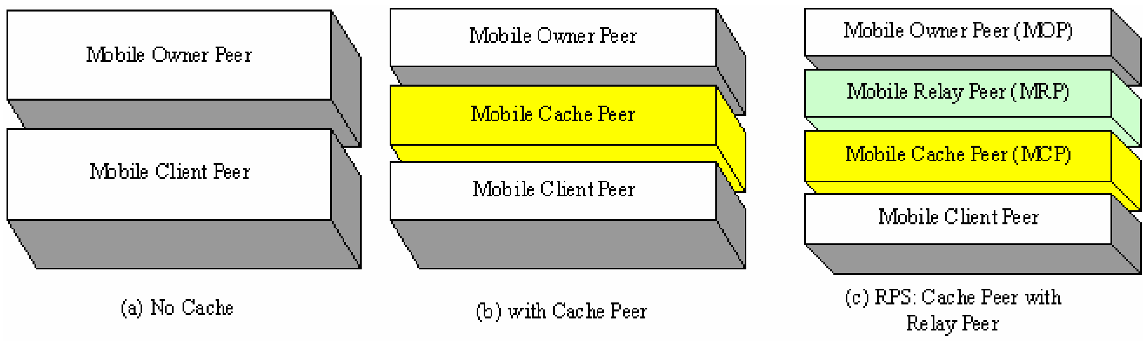


Fig 3. Hierarchy Architecture

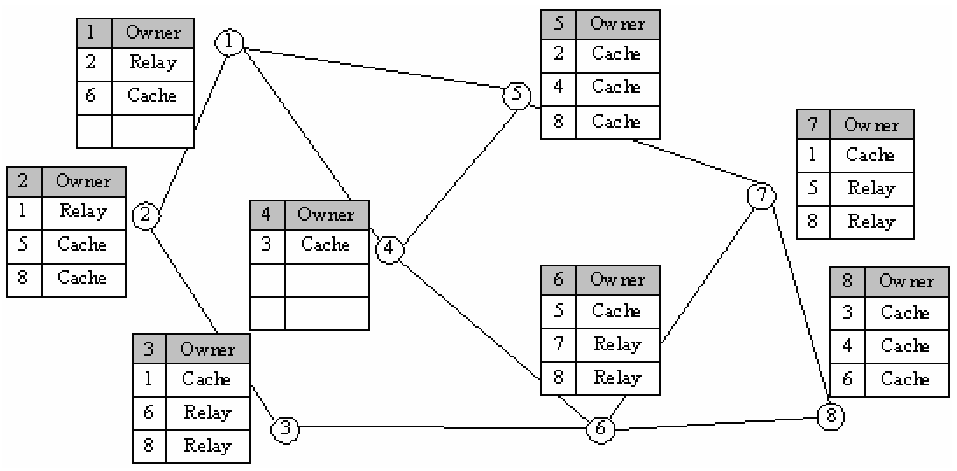


Fig 4. An Example of RPCC

Figure 3 depicts the conventional framework (a), the simple cache framework (b) and our RPCC-based framework(c) As we can see, comparing with the conventional caching approach, in RPCC, a new layer, called the relay peer layer, is created between the source host and the cache node layers. Figure 4 show an example, providing mode details to illustrate the the RPCC approach. Each host serves as the source host for some data item, while at the same time, caches data items from other hosts. It can also serve as a relay peer node for a cached data item if it has high sharing capability.

4.2. Selection of Relay Peers

In RPCC, stable, capable and powerful nodes are chosen as candidates for relay peers. If a candidate can listen to the invalidation messages sent from the source host of its cached data item (i.e., the candidate is within a distance less than a certain number hops from the source host) and successfully negotiate with the source host (i.e., get the approval message from the source host), it can become a relay peer. We use three parameters to define the relay peer selection criterion:

- CAR: Coefficient of peer access rate
- CS: Coefficient of the stability of a node
- CE: Coefficient of the energy level of a node

CAR, CS and CE are calculated every period of time ϕ. To derive CAR, we first define the peer access rate (PAR).

$$PAR = N_a/\phi \qquad (4.2.1)$$

where N_a is the number of cache accesses at a node during ϕ. To learn about the gradual change in PAR over a period of time by utilizing some history information, we record three time windows and define:

$$PAR_t = PAR_{t-2}*\omega/4 + PAR_{t-1}*\omega/2 + N_a/\phi*(1-\omega/4-\omega/2) \qquad (4.2.2)$$

where ω is a parameter to measure the importance of the most recent value in comparison with the history values. To ensure that CAR is always between zero and one, we define as follows:

$$CAR=1/(1+PAR_t) \qquad (4.2.3)$$

Likewise, we define PSR_t as peer switching rate at time t and PMR_t as peer moving rate at time t.

$$PSR_t=PSR_{t-n}*\omega+N_s/\phi*(1-\omega) \qquad (4.2.4)$$

$$PMR_t=PMR_{t-n}*\omega+N_m/\phi*(1-\omega) \qquad (4.2.5)$$

where N_s is the number of times a node has switched its status (reconnected/disconnected) during ϕ and N_m is the number of times a node has moved (from one subnet to another) during ϕ . In fact, peer's state switching and movement will bring the same effect on the system. So we can define CS as

$$CS=1/(1+PSR_t+PMR_t) \qquad (4.2.6)$$

Finally, we define CE as the following:

$$CE=PER_t/E\_MAX \qquad (4.2.7)$$

where PER_t is the peer energy level at time t and E_MAX is the maximum energy level.

We predefine three thresholds μ_{CAR} , μ_{CS} , and μ_{CE} $\in (0,1]$. If the coefficients of a cache node satisfy

$$(\text{CAR}< \mu_{CAR}) \wedge (\text{CS} > \mu_{CS}) \wedge (\text{CE}> \mu_{CE}) \qquad (4.2.8)$$

it means that this cache node is accessible, stable and powerful enough to play the role of a relay peer. Then the node can be chosen to relay peer candidate.

Figure 5 shows the state transition diagram that describes the relay peer formation. If a cache node satisfies the coefficient conditions, it becomes a relay peer candidate. Note that the caching functionality of relay peer candidate is still the same as a cache node. When a relay peer candidate cannot satisfy the coefficient conditions any more, it will switch back to be a normal cache node. A relay peer candidate can send a message to the source host to apply for its promotion to a relay peer. After it receive an approval message from the source host, the candidate becomes a relay peer.

4.3. The Protocol

The RPCC protocol uses some data structures and several types of messages. They are shown in Figure 6(a).

The source host periodically floods out an ***INVALIDATION*** message. Each invalidation message is associated with a TTL (Time To Live) which defines the scope the invalidation message can reach. A node can listen to the invalidation message only when it is within a distance less than TTL number of hops from the source host. The effect of different TTL values on the system performance will be discussed in Section 5 when we describe the results of performance evaluation.

Relay peers can use ***INVALIDATION*** to check whether their cached copies are up-to-date. On the other hand, when receiving ***INVALIDATION***, a relay peer candidate will send an ***APPLY*** message to the source host for its promotion to relay peer. After the source host receives the ***APPLY*** message, it will send back an ***APPLY_ACK*** to the candidate and the candidates becomes a relay peer. When a relay peer switched back to be a normal cache node, it use ***CANCEL*** to inform the source host.

If the data item has been modified, the source host sends a ***UPDATE*** message to its relay peers to inform them of the update.

If a cache node cannot decide whether its cached copy is consistent with the master copy, it has to poll the relay peer with a ***POLL*** message. When the relay peer receives the ***POLL***, it first checks the status of its own cached copy. If it the copy is of the newest version, it sends back a ***POLL_ACK_A*** message (if the copy at the cache node is up-to-date) or ***POLL_ACK_B*** (if the copy at the cache node is stale); otherwise, the relay peer has to wait for the next ***INVALIDATION*** to confirm the status of its cached copy.

GET_NEW and ***SEND_NEW*** are used to invalidate the cached data between the relay peer and the source host when a relay peer reconnects to the network finding that its data item has already been updated.

The formal descriptions of the algorithms used by a source host, relay peer, and cache node are given in Fig 6(b), 6(c) and 6(d), respectively.

4.4. Meeting Different Consistency Requirements

RPCC can provide query requests with different levels of consistency guarantees.

If only weak-consistency is needed, the request can be served immediately by a cache node. If the request needs the Δ-consistency guarantee and the TTP is larger than zero, the request can also be answered immediately (in RPCC, TTP is the Δ value). If TTP equals to zero or strong consistency is required, the cache node will poll the relay peer to check the consistency of the data. If the TTR of the cached copy at the relay peer is larger than zero, which indicates that the data at the relay peer is up-to-date, then the cache node can compare the versions and answer the cache peer. Otherwise, it will wait for the next invalidation message from the source host.

4.5. Handling Disconnection/ Reconnection

In a MP2P system, a mobile host may disconnect from and/or reconnect to the wireless network from time to time without giving any notice. Disconnection/reconnection is one of the most important factors that complicate the cache invalidation problem. For a strategy to be effective and practical, it should be resilient to the problem.

First, let us discuss how RPCC handles the failure of an source host. If the source peer fails, cache peers can not receive the ***INVALIDATION*** and ***UPDATE*** thus can not know the status of the data item at the source host. Strong consistency can be ensured only for TTR_d time. If the source host misses the ***GET_NEW*** message from a relay peer and reconnects before the next broadcast interval, the relay peer will wait for the next ***INVALIDATION*** message and re-send ***GET_NEW***. Similarly, if the source host misses the ***APPLY*** message from the relay peer candidate, the candidate loses an opportunity to switch to a relay peer and need to wait for the next switching period.

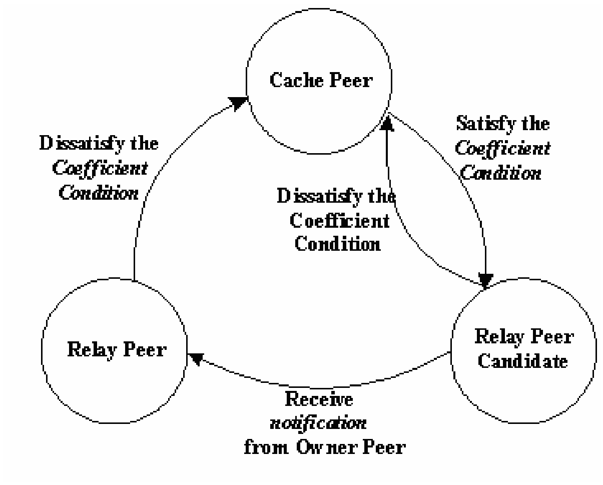

Cache Peer
Dissatisfy the Coefficient Condition
Satisfy the Coefficient Condition
Dissatisfy the Coefficient Condition
Relay Peer
Relay Peer Candidate
Receive notification from Owner Peer

Fig 5. The State Diagram

Data Structures:

ID_i: the id of data item, $ID_i \in D$
OP_i: the owner peer of data item, $OP_i \in M$
RP_i: the relay peer of data item, $RP_i \in M$
CP_i: the cache peer of data item, $CP_i \in M$
VER_i: the version of data item
CT_i: the content of data item
CL_i: consistency level (strong, delta, weak)
TTN_i: time to notify the relay peers (for data on the owner peer)
TTP_i: time to poll the relay peers (for data on the cache peer)
TTR_i: time to refresh (for data on the relay peer)
$LVER_i$: the version of local data item
Source data: $< ID_i, CT_i, CL_i, VER_i, RP_i, TTN_i >$
Relay data: $< ID_i, CT_i, CL_i, VER_i, TTR_i >$
Cache data: $< ID_i, CT_i, CL_i, VER_i, TTP_i >$

Message Types:

UPTATE(ID_i, OP_i RP_i, CT_i, VER_i)
INVALIDATION(ID_i, OP_i VER_i)
GET_NEW(ID_i, OP_i RP_i)
SEND_NEW(ID_i, RP_i, CT_i, VER_i)
APPLY(ID_i, OP_i RP_i)
APPLY_ACK(ID_i, OP_i, RP_i)
CANCEL(ID_i, OP_i RP_i)
POLL (ID_i, CP_i VER_i)
POLL_ACK_A (ID_i, CP_i, VER_i)
POLL_ACL_B(ID_i, CP_i, VER_i, CT_i)

Fig 6(a). Data Structures and Message Types

```
(1)  IF $TTN_i$ =0                  //at invalidation interval
(2)      IF data item is updated during this period
(3)          For all peers $\in RP_i$;
(4)          Send UPDATE to these relay peers ;
(5)      END IF
(6)      Broadcast INVALIDATION;
(7)      Renew $TTN_i$;
(8)  END IF

(9)  IF receive GET_NEW          //relay peer misses the last update message
(10)     Send back SEND_NEW to such kind of relay peers;
(11) END IF

(12) IF receive APPLY            //candidate wants to promote to relay peer
(13)     Send back APPLY_ACK;
(14)     Add this peer to its relay peer table ;
(15) END IF

(16) IF receive CANCEL or the destination peer of APPLY_ACK unreachable
(17)     Remove the peer from its $RP_i$;
(18) END IF
```

Fig 6(b). The Protocol: Source Peer Side

```
(1)  IF receive INVALIDATION          //from owner peer
(2)    Compare its $LVER_{ij}$ with $VER_{ij}$ ;
(3)        IF $LVER_{ij} < VER_{ij}$          //not up-to-date
(4)            Send GET_NEW to owner peer for the latest version ;
(5)        ELSE renew $TTR_{ij}$;
(6)        END IF
(7)  END IF

(8)  IF receive POLL                  //from cache peer
(9)      IF $TTR_{ij}>0$                 //up-to-date (at relay peer)
(10)         Compare the $LVER_{ij}$ with $VER_{ij}$;
(11)         IF $LVER_{ij} = VER_{ij}$        //up-to-date (at cache peer)
(12)             Send back POLL_ACK_A;
(13)         ELSE                        //stale (at cache peer)
(14)             Send back POLL_ACK_B;
(15)         END IF
(16)     ELSE                            //stale (at relay peer)
(17)         Wait for the INVALIDATION message;
(18)     END IF
(19) END IF

(19) IF receive SEND_NEW              //from owner peer
(20)      Update the data item ;
(21)      Renew $TTR_{ij}$;
(22) END IF

(23) IF receive UPDATE                //from owner peer
(24)      Update the data item ;
(25) END IF
```

Fig 6(c). The Protocol: Relay Peer Side

```
(1)  IF receive query request
(2)    IF ($CL_i$=揤eak�)
(3)          Answer the query immediately ;
(4)    ELSE
(5)      IF ($CL_i$=揹elta? and $TTP_i>0$)
(6)          Answer the query immediately ;
(7)      ELSE
(8)          Broadcast POLL;
(9)      END IF
(10)   END IF
(11) END IF

(12) IF receive POLL_ACK_A            //unchanged
(13)     Answer the query immediately ;
(14)     Renew $TTP_i$;
(15) END IF

(16) IF receive POLL_ACK_B            //changed
(17)     Update the data item ;
(18)     Answer the query;
(19)     Renew $TTP_i$;
(20) END IF

(21) IF want to be a relay peer       //candidates
(22)    Send APPLY to owner peer;
(23) END IF

(24) IF receive APPLY_ACK             //acknowledgment from owner peer
(25)   Change its status from cache peer to relay peer ;
(26) END IF

(27) IF receive UPDATE                //from owner peer
(28)    IF relay peer candidate       //it misses the APPLY_ACK
(29)      Change its status to relay peer ;
(30)      Update its data item ;
(31)      Renew $TTP_i$;
(32)    Else                          //the owner peer misses the CANCEL
(33)      Update its data item ;
(34)      Send CANCEL to owner peer;
(35)      Renew $TTP_i$;
(36)    END IF
(37) END IF
```

Fig 6(d). The Protocol: Cache Peer Side

Next, we describe how RPCC handles the failure of a relay peer. A relay peer may disconnect from the network for a while and miss some ***UPDATE*** messages. During the periods, the data item at the source host may have already been changed. Here, we use VER_d in ***INVALIDATION*** to mark the version of the data item. When the relay peer
receives the message, it compares VER_d in the message with that of its local cached copy ($LVER_d$). If $LVER_d < VER_d$, which indicates that the data item has been updated when the relay peer was disconnected, the peer will send a ***GET_NEW*** message to the source host and get the most up-to-date version of the data item. Otherwise, the data has not been updated during the disconnection period and the reconnected relay peer can just renew its TTR_d.

Finally, we need to deal with cache node failure. The case that the cache node is out of work after it receives the query request from one client peer is beyond the scope of our discussion. Here, we consider the situation that the cache node is not reachable (it may be disconnected from the network or move away) after it sends an ***APPLY*** message to the source host and has not got the ***APPLY_ACK*** yet. At this moment of time, the source host has already added this cache node to its *relay peer table* but the candidate peer can not receive the ***APPLY_ACK***. This kind of disconnection can be discovered in the MAC layer. The source host will remove the peer from its relay peer table and will not send ***UPDATE*** message to it.

5. Performance Evaluation

We have carried out a comparative performance study of the proposed RPCC approach through simulations. We compared the performance of RPCC with both push and pull based strategies. In this section, we report the results of the performance evaluation. We first describe out simulation environment. Then,, in Subsection 5.1, we demonstrate the effectiveness of our RPCC on reducing the network communication overhead. In Subsection 5.2, we demonstratesthe effectiveness of the proposed strategy on reducing the query latency. The impact of invalidation message's TTL on the system performance under RPCC is studied in subsection 5.3.

We conducted the simulation on the GloMoSim [Zen98] simulator. We carried out simulations with 50 mobile peers in a size 1500*1500 flatland. The used movement pattern of mobile peers is random-waypoint [Joh96]. Each mobile host generates an independent stream of updates to its source data and its query requests with an exponentially distributed update interval and an exponentially distributed query interval. The processing time by a node is considered to be negligible. The parameters used in the simulation are listed in Table 1.

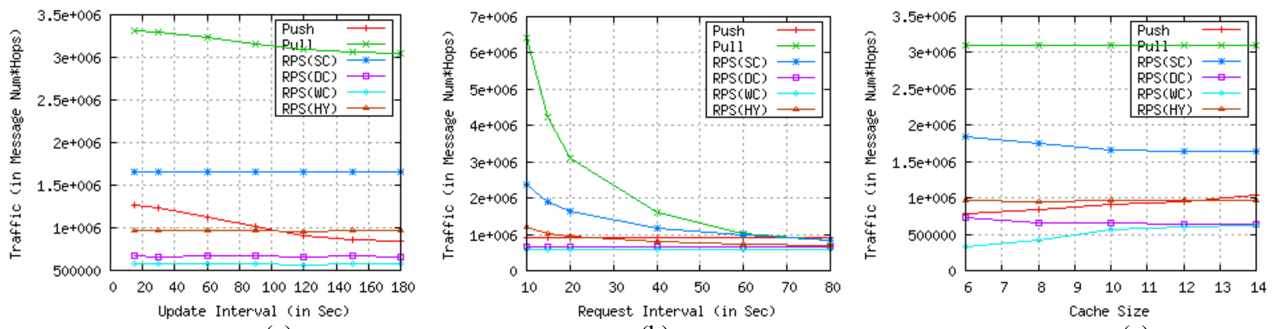
Table 1. Simulation Parameters

Parameter	Description	Default Value
N_Peers	Number of peers in the network	50
T_Area	Physical terrain dimension of the network	1.5km*1.5km
C_Num	Cache Number of each mobile host	10
C_Range	Communication range of mobile hosts	250m
T_Sim	Simulation time	5 hours
I_Update	Average interval of data item update	2 minutes
I_Query	Average interval of query requests	20 seconds
TTL_BR	TTL of broadcast message in simple push/pull	8 hops
TTL_I_RPS	TTL of invalidation message in RPS	3 hops
TTN_OP	TTN of data item at owner peer	2 minutes
TTR_RP	TTR of data item at relay peer	1.5 minutes
TTP_CP	TTP of data item at cache peer	4 minutes
I_Switch	Switching interval of each peer	5 minutes
μ_{CAR}	Threshold of CAR, see Eq4.2.3	0.15
μ_{CS}	Threshold of CS, see Eq4.2.6	0.6
μ_{CE}	Threshold of CE, see Eq4.2.7	0.6
ω	Weighting parameter of recent/history values	0.2

5.1. Network Traffic

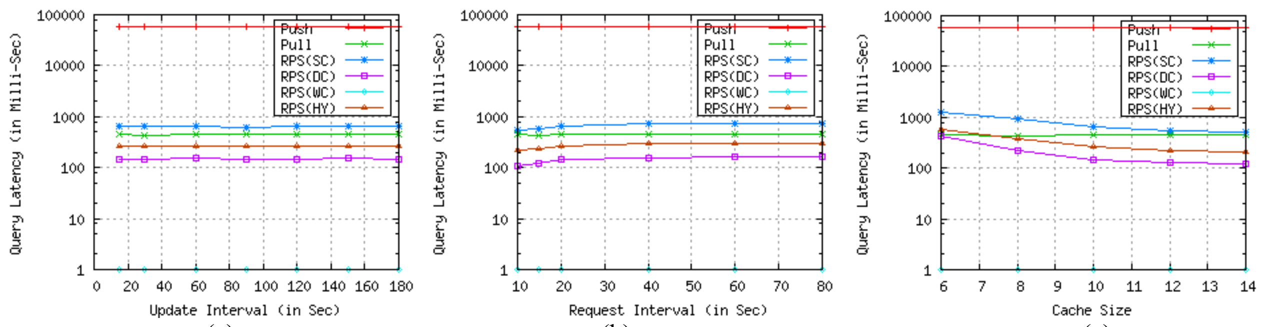
Fewer communication messages lead to less network traffic and energy consumption. Therefore, a good algorithm should send less number of messages. Fig 7 shows the performance results in terms of network traffic as a function of the update interval, request interval and cache number for different strategies, respectively. As we can see, the pull based strategy always leads to a heavy communication overhead. This follows the fact that in the pull based strategy, each time when a query request comes, the cache node to poll the source host to invalidate the status of the data items it caches. However, in the push based strategy, during each invalidation period, the data update information can be pushed to all the cache nodes in one time, leading to the reduction in the network traffic. RPCC is a hybrid scheme where both push and pull are performed. With the introduction of relay peers, the TTL value for the flooding invalidation messages can be made much smaller. Therefore, the network traffic can be decreased greatly. Moreover, RPCC can deal with three kinds of consistency requirements simultaneously.

Fig 7(a) shows the comparison under different update intervals. The pull based strategy produces far more traffic than other strategies. RPCC with delta consistency (DC) and weak consistency (WC) only need fewer communications because the cache node can answer most query requests immediately. The traffic caused by RPCC with strong consistency (SC) is bit higher due to the polling between cache nodes and the relay peers. But it still saves more messages than the pure pull strategy. In a more practical hybrid scenario that requests with three different consistency requirements come with the same probability (HY), the network traffic caused by RPCC is nearly at the same the level as push strategy.



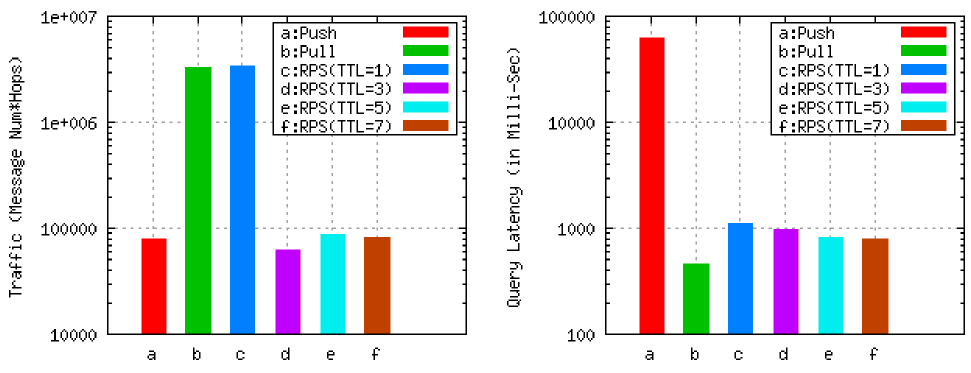
(a) (b) (c)

Fig 7. Network Traffic



(a) (b) (c)

Fig 8. Query Latency



(a) Traffic (b) Query Latency

Fig 9. Impact of TTL

Fig7(b) suggests that when the request interval enlarges, more flooding polling messages in pull based strategy can be saved thus the communication overhead can be decreased dramatically, but it is still higher than RPCC and push based strategy.

Fig 7(c) shows the traffic when the cache number changes. We can observe that cache size has little impact on the traffic of pull based strategy because the consistency checking is initiated by query requests. Given the fixed query frequency, the communication overhead is stable. But in a push based strategy, a larger cache size leads to more cache nodes thus produces more data item updates between source peers and cache peers. In RPCC, more cache peers can also bring more relay peers. So, the pull traffic can reduce while the push traffic increases at the same time.

5.2. Query Latency

Figure 8 evaluates the performance in terms of query latency (in log scale). In the push based strategy, a cache node has to wait for the invalidation messages from the source host. So the average query latency is longer than half of the invalidation interval. While in the pull based strategy, the on-demand polling can decrease the query latency greatly comparing with the push type. In our RPCC, the push between source host and relay peers and the pull between relay peers and cache nodes are performed simultaneously, thus the query latency is short. Moreover, in RPCC, the checking hops are much fewer than the simple push and/or pull based strategies, which can reduce query latency farther.

Fig 8(a), (b) and (c) show the results under different update interval, request interval and cache number, respectively. We note that the query latency of RPCC is at the same level as pull based strategy and is much shorter than that of push based strategy. The results are consistent with our expectation. Fig 8(c) also implies the fact that, with the increase of cache size, the query latency of RPCC decreases. This is because of the augment of the number of relay peers, which helps to answer the query requests without any delay.

5.3. Impact of TTL for Invalidation Messages on System Performance

In RPCC, if there are more relay peers in the network, each cache node can easily find a nearby relay peer to invalidate the data items it caches. But the cost of communications between each source host and its relay peers is much higher because a larger number of relay peers will require the data update from the source peer. On the contrary, fewer relay peers can reduce the network traffic between source host and relay peers but degrade the invalidation efficiency of relay peers because it will not be so easy for each cache node to find a relay peer for polling. Since the TTL value associated with the invalidation message sets the scope the invalidation message can reach, a larger TTL can give more cache nodes the opportunity to become relay peers. The impact of TTL value on the system performance is shown in Fig 9.

We vary the TTL from 1 to 7 hops. As a reference, we also simulated the simple push and pull strategies. Here we only discuss the RPCC(SC) case that has the same consistency requirement as push and/or pull strategies. We assume one peer is randomly selected as the source host and its data item is cached by all other peers. Other parameters are the same as used in previous simulations. In Fig 9(a), we observe that when TTL of invalidation message in RPCC is 1, the network traffic is similar to simple pull strategy. This is because, in this situation, only few cache nodes one-hop from the source peer can be relay peers thus RPCC is more like the simple pull based strategy; vice versa, larger TTL value (e.g. TTL=7) can bring more relay peers and RPCC is more like the simple push based strategy. Moreover, with more relay peers, most query requests can be answered immediately thus the query latency can be reduced to some extent. Fig 9(b) compares the query latency for different strategies.

6. Conclusion

In this paper, we have studied the design of efficient cache consistency invalidation strategies for cooperative caching in a MP2P system over a MANET. We have identified the issues that need to be considered and presented a novel relay peer based strategy (RPCC) for mobile peer-to-peer environments. The key feature of RPCC is that we choose some appropriate peers as the relay peers to transfer the data update information. This feature makes the proposed RPCC highly flexible and efficient. Moreover, RPCC can adapt to three different kinds of consistency requirements simultaneously. We evaluated the performance of RPCC with comparison to push and pull based strategies in terms of communication overhead and query latency. The results showed that RPCC can reduce the network traffic significantly comparing with pull based strategy without affecting the query delay as well. We also showed that the TTL value of invalidation message in RPCC has impact on the performance in the sense that, on one extreme, if the TTL value is small so that too few relay peers are available, the system is more like a simple pull based system; on the other extreme, if the TTL value is larger enough that most of the cache peers can be relay peers, the system will be more like a simple push based system.

In the future, we plan to extend out work in the following three directions. First, both our RPCC and traditional simple push/pull strategies need to pre-set the push/pull frequency. But under realistic environments, the frequency of query requests changes dynamically. We plan to investigate how to change the push/pull frequency adaptively according to the runtime system conditions thus make RPCC more flexible and efficient. Second, the number of relay peers is important to the performance of RPCC. In current strategy, the number of relay peer cannot be controlled. We will evaluate the effect of the relay peer number on the system performance in a more detailed way and to find an effective strategy to control the relay peer overlay formation. Third, we plan to explore strategies that provide consistency of replicas in MP2P network. Unlike cache that master copy can only be updated by its source peer, as to replicas, any peer that has the replica can modify the data, which make the consistency maintenance more complicated. We would like to see how to address such issues.

Acknowledgement

This work is partially supported by University Grant Council of Hong Kong under the CERG grant B-Q822 (PolyU 5183/04E), the Hong Kong Polytechnic University under the ICRG grant A-PF77, and the 973 Program of China under the grant 2002CB312002.

References

[Bar94] D. Barbara and T. Imielinksi, "Sleepers and workaholics: Caching Strategies for Mobile Environments", ACM SIGMOD pp.1-12, 1994.

[Cao02] G. Cao, "On improving the Performance of Cache Invalidation in Mobile Environments", ACM/Baltzer Mobile Networks and Application (MONET), vol. 7, no. 4, pp. 291--303, Aug. 2002.

[Cao03] G. Cao, "A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments", ACM MOBICOM'00, pp. 200-209, Aug. 2000.

[Cao04] G. Cao, L. Yin and C. R. Das, "Cooperative Cache-based Data Access in Ad Hoc Networks", IEEE Computer Magagine, Vol.37, No.2, pp. 32-39, 2004.

[Cho04] C. Y. Chow, H. V. Leong, A. Chan, "Peer-to-Peer Cooperative Caching in Mobile Environments", in Proc. of 24th International Conference on Distributed Computing Systems Workshops (ICDCSW'04), 2004

[Cor99] Corson, M. and J. Macker, "Mobile Ad Hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations", RFC 2501 January 1999.

[Din04] G. Ding and B. Bhargava, "Peer-to-Peer File Sharing over Mobile Ad Hoc Networks", In Proc. of IEEE Annual Conference on Pervasive Computing and Communications Workshops, pp.104– 109, March 2004.

[Gra89] C. Gray and D. Cheriton, "Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency", in Proc. of the 12th ACM Symposium on Operating Systems Principles, 1989.

[Gwe96] J. Gwertzman and M. Sltzer, "World-Wide Web Cache Consistency", in Proc. of 1996 USENIX Tech. Conference, pp. 141-151, San Diego, CA, 1996.

[Har01] T. Hara, "Effective Replica Allocation in Ad Hoc Networks for Improving Data Accessibility", in Proc. of IEEE Infocom 2001, pp.1568-1576, 2001.

[Har02a] T. Hara, "Replica Allocation in Ad Hoc Networks with Periodic Data Update", in Proc. of the Third International Conference on Mobile Data Management, pp.79-86, January, 2002.

[Har02b] T. Hara, "Cooperative Caching by Mobile Clients in Push-based Information Systems", in Proc. of ACM Int'l Conf. on Information and Knowledge Management (ACM CIKM'02), pp. 186-193, Nov. 2002.

[How88] J. Howard, M. Kazar, et al, "Scale and Performance in a Distributed File System", ACM Transactions on Computer Systems, Vol. 6, No. 1, 1988.

[Jin 97] J. Jing, A. Elmagarmid, A. Helal and R. Alonso, "Bit-Sequences: An Adaptive Cache Invalidation Method in Mobile Client/Server Environments", ACM-Baltzer Journal on Special Topics in Mobile Networks and Applications (MONET), Vol. 2, No. 2, pp115-127, 1997.

[Joh 96] D.B. Johnson and D.A Maltz, "The dynamic source routing protocol for mobile ad-hoc networks", Mobile Computing, Imielinksi and H. Korth.Eds. Kluwer., pp 153–181, 1996.

[Kah01] A. Kahol, S. Khurana, et al, "A strategy to manage cache consistency in a distributed mobile wireless environment", in Proc. of International Conference of Distributed Computing Systems, pp. 530--537, 2000.

[Lai03] K. Y. Lai, Z. Tari, P. Bertok, "Cost Efficient Broadcast Based Cache Invalidation for Mobile Environments", in Proc. of the 2003 ACM symposium on Applied computing, pp. 871-877, 2003.

[Lan03] J. Lan, X. Liu, P. Shenoy and K. Ramamritham, "Consistency Maintenance in Peer to Peer File Sharing Networks", in Proc. of IEEE Workshop on Internet Applications (WIAPP), San Jose, CA, pp. 90-94, 2003

[Lim04] S. Lim, W. Lee, G. Cao and C. Das, "Performance Comparison of Cache Invalidation Strategies for Internet-based Mobile Ad Hoc Networks", IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS), 2004.

[Liu97] C. Liu and P. Cao, "Maintaining Strong Consistency in the World-Wide Web", In Proc. of ICDCS, 1997.

[Mil02] D. S. Milojicic, V. Kalogeraki, et al, "Peer to Peer Computing", *Technical Report: HPL-2002-57*, 2002.

[Nel88] M. Nelson, B. Welch and J. Ousterhout, "Caching in the Sprite Network File System", ACM Transactions on Computer Systems, 6(1):134--154, February 1988.

[Wan04] Z. Wang, S. Das, H. Che and M. Kumar, "SACCS: Scalable Asynchronous Cache Consistency Scheme for Mobile Environments", IEEE ICDCS: International Workshop on Mobile and Wireless Networks (MWN), Rhode Island, pp797-802, May 2003

[Wu96] K. Wu, P. Yu and M. Chen, "Energy-Efficient Caching for Wireless Mobile Computing", in Proc. of the Twelfth International Conference on Data Engineering, pp.336-343, 1996.

[Yin99] J. Yin, L. Alvisi, M. Dahlin, C. Lin, "Volume Leases for Consistency in Large-Scale Systems", IEEE Transactions on Knowledge and Data Engineering, v.11 n.4, p.563-576, July 1999.

[Zen98] X. Zeng, R. Bagrodia, M. Gerla, "GloMoSim: a Library for Parallel Simulation of Large-scale Wireless Networks", Proc. of the 12th Workshop on Parallel and Distributed Simulations -- PADS '98, May 26-29, 1998

[Zhe04] J. Zheng, Y. Wang, X. Lu and K. Yang, "A Dynamic Adaptive Replica Allocation Algorithm in Mobile Ad Hoc Networks", In Proc. of IEEE Annual Conference on Pervasive Computing and Communications Workshops, pp.65-69, March 2004.