Computer Communications 32 (2009) 41–50

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

A peer-to-peer file search and download protocol for wireless ad-hoc networks ☆

Hasan Sözer [a,*], Metin Tekkalmaz [b], Ibrahim Korpeoglu [b]

[a] Faculty of Electrical Engineering, Mathematics and Informatics, University of Twente, Enschede, The Netherlands
[b] Department of Computer Engineering, Bilkent University, Ankara, Turkey

ARTICLE INFO

Article history:
Received 15 March 2007
Received in revised form 26 August 2008
Accepted 4 September 2008
Available online 11 September 2008

Keywords:
Wireless ad-hoc networks
Peer-to-peer networks
File sharing

ABSTRACT

Deployment of traditional peer-to-peer file sharing systems on a wireless ad-hoc network introduces several challenges. Information and workload distribution as well as routing are major problems for members of a wireless ad-hoc network, which are only aware of their immediate neighborhood. In this paper, we propose a file sharing system that is able to answer location queries, and also discover and maintain the routing information that is used to transfer files from a source peer to another peer. We present a cross-layer design, where the lookup and routing functionality are unified. The system works according to peer-to-peer principles, distributes the location information of the shared files among the members of the network. The paper includes a sample scenario to make the operations of the system clearer. The performance of the system is evaluated using simulation results and analysis is provided for comparing our approach with a flooding-based, unstructured approach.

1. Introduction

Peer-to-peer networks have been very popular since their first emergence. Several peer-to-peer file sharing systems have been deployed and are functional on the Internet, such as Napster [1], Gnutella [2] and FastTrack [3]. Similar systems currently serve many users who are able to share files located on their personal computers. Together with the new users of the Internet and the emergence of different types of files to be shared (documents, audio/video files, etc.), the number of users of peer-to-peer file sharing systems increases every day.

In the meantime, mobile devices and wireless communication technologies have evolved and become very popular. Both areas have experienced rapid improvements during the last few years, which have led to the development of high-performance products. Today, personal digital assistants (PDAs) have almost the same abilities of ordinary desktop computers despite their small size and weight. On the other hand, new wireless technologies enable handheld devices to communicate and form ad-hoc networks easily and automatically. Bluetooth [4], for instance, is one such technology that uses short-range radio communication and interconnects handheld electronic devices ranging from cellular phones to PDAs.

Today, high-performance handheld devices can communicate with each other in a wireless ad-hoc network (WANET). Such an environment provides the possibility to share files. Moreover, peer-to-peer systems that are often employed for file sharing are also suitable for WANETs, since they do not require any infrastructure. However, the deployment of traditional peer-to-peer file sharing systems on a WANET introduces several challenges. Such networks can be formed anytime/anywhere without requiring any infrastructure, and the nodes of the network may change their locations. In addition, a peer-to-peer file sharing system that is running on the Internet relies on the network layer (IP) for communication between nodes and for downloading files. A WANET needs to run an ad-hoc routing algorithm to provide these services. Several protocols [5,6] have been proposed to route packets in a WANET, and some have been standardized [7]. However, we still lack a widely accepted common routing protocol that is implemented and deployed. Moreover, the standardization efforts tend to keep the WANET routing protocols simple (due to the heterogeneity of mobile systems), which can be inadequate to provide the services needed for specific applications [8].

The work presented in this paper takes a cross-layer design approach where the lookup functionality and the routing functionality are unified. We propose a file sharing system, which determines both *from where* and *how* to obtain a file in a WANET. The system works in a peer-to-peer manner and it distributes the location information of the shared files among the members of the network. To store and maintain the location information together with the routing information, the system uses a distributed hash table and a tree-structure based on the topology of the network. The system also employs and adapts dynamic source routing [5] and peer-to-peer location lookup techniques [9,10].

We present a sample scenario to describe how the set of system operations work together to accomplish file sharing among the

☆ This work is partially supported by TUBITAK The Scientific and Technological Research Council of Turkey (Project n. EEEAG 104E028), and by the European Commission in the framework of the FP7 Network of Excellence in Wireless COMmunications NEWCOM++ (contract n. 216715).

* Corresponding author. Tel.: +31 53 489 5682; fax: +31 53 489 3247.
E-mail addresses: sozerh@ewi.utwente.nl (H. Sözer), metint@cs.bilkent.edu.tr (M. Tekkalmaz), korpe@cs.bilkent.edu.tr (I. Korpeoglu).

0140-3664/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2008.09.004

members of a WANET. We also provide the simulation results of the system and we elaborate on the messaging requirements to maintain the distributed index, to perform file queries, and to access files. Simulation results showed that the high mobility of the nodes leads to poor performance due to frequent updates of the distributed index. For this reason, we also make a trade-off analysis and we compare the bandwidth efficiency of our system with that of a flooding-based, unstructured approach. Results showed that our system performs much better as the network size increases. In a WANET with 100 nodes, less than one file query per node is enough to amortize the cost of an index update.

The remainder of this paper is organized as follows. In the next section, related previous studies are summarized. In Section 3, an overview of the system is given, which is followed in Section 4 by a detailed description of each operation supported by the system. In Section 5, a working scenario of the system is described to show how each operation updates and maintains the distributed location and routing information stored in the system. Next, in Section 6, simulation results and discussions on them are presented. Finally, in Section 7 conclusions are given and some future work issues are discussed.

2. Related work

As far as peer-to-peer (P2P) file sharing is considered, Napster [1] appears to be one of the earliest and most popular applications. Napster, in its initial form, enabled file sharing among computers on the Internet, which is by nature unpredictable, since it is hard to predict when the computers connect and disconnect. The main idea behind Napster is a central server that stores index information (i.e. filename and address pairs), which is used to answer queries about where the files are stored on the Internet. Once the location of a file is determined, file transfers are carried out in a P2P manner. Although the actual file transfers are P2P, index information is accessed using a client–server paradigm. Napster enables easy location lookup by using a central server, but it is affected by the typical weaknesses of centralized systems. Several studies have been carried out in recent years to cope with problems posed for P2P file sharing by the dense, highly dynamic and lowly aware nature of the Internet. More recent works aim for fully distributed P2P systems; therefore, they store index information in a distributed manner. One such distributed system, Content-Addressable Network (CAN) [9], is based on a fully distributed hash table. In CAN, filenames are hashed and mapped to points in a d-dimensional space. The d-dimensional space is divided into chunks and distributed among the members of the network where each member is responsible for one portion of the space (i.e. a chunk). Along with a chunk, each node stores some information about the neighboring nodes, which makes searching of files possible by providing the location information for files and an overlay network-level routing. Chord [11] is another well-known fully distributed P2P system in which a ring shaped overlay network is applied. Each node on this ring maintains pointers to other nodes at various distances. To gather the location information of a file, these pointers are followed in a manner that shortens the access path as much as possible. Ref. [10] surveys P2P content distribution technologies and it can be referenced for other significant P2P file sharing methods.

The P2P file sharing approaches mentioned so far are mainly designed for the Internet. The wireless ad-hoc networks (WANETs) counterpart of the same problem comes with several difficulties due to the dynamic nature of the WANETs as stated previously. The first work on P2P file sharing on WANETs is 7DS [12]. 7DS allows nodes with an intermittent Internet connection to browse the web, in which, whenever a node fails to connect to the Internet, it can search for the required data among its peers. Ref. [13] is based on partial flooding where the searches are carried by queries broadcasted several hops ahead, and where flooding the entire network is prevented by mechanisms like caching and selective routing. ORION, described in [14] and being another P2P file sharing approach for WANETs, also employs flooding for the file queries. The query results are returned selectively, hence duplicate results are avoided for the same file. During the file transfer phase, the caches constructed while querying the files are used for routing. A more recent work on ad-hoc P2P file sharing, which is described in [15], is also based on flooding. Various heuristics and techniques like replication, query filtering and limiting the number of hops for message forwarding are applied in order to reduce the overhead of the flooding. [16] employs context-awareness toward the same aim. Approaches based on flooding work fine for small WANETs but as the network gets larger they cause traffic overhead and the probability of finding a file in the network reduces. In our work, rather than utilizing a flooding-based protocol and introducing techniques to reduce its overhead, we propose a novel, cross-layer system, which combines a location information service (see [17] for more information on location information services) and routing functionality. Our system is designed and specialized to provide a deterministic way to locate and access files (i.e. if a file is shared in the WANET, its location can be determined and it can be accessed).

Virtual ring routing (VRR), proposed in [18], is one of the most recent studies on routing which can successfully work on WANETs. Like many overlay routing protocols, VRR employs distributed hash tables (DHT), but it is directly implemented over the link layer and thus does not require an underlying network routing protocol. VRR performs better than many existing routing protocols working on WANETs since it requires neither network flooding nor translation between fixed identifiers and location-dependent addresses. Although VRR is proposed as a routing protocol, it can also provide DHT functionality in which keys can identify application objects (e.g. file locations) instead of routes to the nodes. But in order to provide this functionality, VRR requires additional messaging, whereas our algorithm employs a cross-layer approach, hence distributing the key-value pairs is enough both for DHT and routing functionality.

Both our algorithm and VRR make use of overlay networks, although they have different designs. The overlay network used by our algorithm is actually a spanning tree of the graph representing the connectivity of the nodes in the network, hence if two nodes are adjacent in the overlay network, they are in the communication range of each other. On the other hand, adjacent nodes in the virtual ring of VRR are not necessarily so in the physical topology of the network. Since the messages are routed through the overlay tree in our algorithm, the paths between the nodes are not always the shortest ones on the actual topology. However our design provides significant advantages over VRR as far as the space and time requirements as well as the maintenance costs of the routing tables are considered. As described in [18], VRR requires $rp + k$ routing table entries per node on the average, where r is the number of virtual neighbors, p is the average path length, and k is the number of physical neighbors. On the other hand, the number of routing table entries of the proposed algorithm is at most k. If p is assumed to grow with $\sqrt{n}$ as in [18], where n is the number of nodes in the network, and k is assumed to be constant for a given node density, the routing table size is $O(r\sqrt{n})$ for VRR, whereas it is constant, that is $O(1)$, for the proposed algorithm. Hence, VRR requires much more space to store the routing tables and processing power to search on the tables for each packet routed. Furthermore, the proposed algorithm requires much less communication overhead to construct the routing tables since the newly connected node only communicates with its parent in the overlay tree, which is in its communication range. On the other hand, in VRR, whenever a new node connects, multihop communication is required between the new node and all of its virtual neighbors in order to update the routing tables and DHT to reflect the new topology.

3. System overview

The system expects three basic functionalities from the underlying network layers:

- Device discovery
- Communication with nodes in the range
- Notification of link failure

Together with these functions the system makes use of a fully distributed hash table where keys are the names of the files to be shared and the values are the globally unique locations of these files (using MAC address of the device and the full path of the file on the device is one way to provide this uniqueness) together with necessary routing information which will be described soon. The basic dynamics of the system are as follows. A one-dimensional space (i.e. a line) is used to store (key, value) pairs by mapping each key to a point P on the "*hashline*" using a uniform hash function. In fact, any hash function that can map a file name to a real number between 0 and 1 may be used for this purpose. However, a function that guarantees uniformity would lead to a more balanced information distribution among the nodes. Each node in the WANET is responsible for storing a segment of the *hashline* (i.e. the hash table entries which correspond to points that are included in this *hashline* segment).

We call the node which is responsible for the segment of the *hashline* containing the point P the *P-Node*, and the node which stores a file with name F the *F-Node*. Hence a *P-Node* stores index information along with location information and an *F-Node* stores the actual file. A file is accessed following the main steps listed below:

(1) The name of the file to be searched is hashed to determine the point P on the *hashline*.
(2) The *P-Node* is accessed.
(3) The location of the searched file (i.e. *F-node*) and the route to that location is determined from the *P-Node*.
(4) The *F-Node* is accessed, and the file is downloaded.

Determining the routes between required nodes is the core and distinguishing part of the system. The system is designed to cope with this problem using a logical tree-structure that is imposed on the nodes of a WANET. The tree-structure helps in accessing to *P-Node*, and the information obtained from the *P-Node* helps in determining the route to the *F-Node* from where the file will be downloaded. Hence, although the network may include loops at the link layer, loops are not allowed in the layer at which the P2P system is implemented, which is usually the application layer. While the network grows with the addition of new members, a new member node is not permitted to join the same file sharing enabled WANET via more than one link (i.e. via more than one neighboring node). A loop-free network can be achieved by providing a unique network ID (e.g. MAC address of the root node) for each file sharing enabled WANET and not allowing a node to have more than one parent with the same network ID.

Fig. 1 depicts an example WANET with six nodes. In this network, let's say Z searches for a file F. This file is stored by X. So, X is the *F-Node*. The file F is mapped to a point P on the hashline and this point falls into the hashline segment that is under the responsibility of C. So, C is the *P-Node* and it stores the route from itself to the *F-Node*, X.

To be able to reach to the file F, Z first reaches to C. Then it obtains the route information to X and downloads the file. The tree-structure network is used for reaching to the *P-Node* since every parent knows the hashline segments of itself and its children. For example, Z knows that the point P does not fall into its segment and it has no children. So, the query is forwarded to its parent, A. When A gets this query, it forwards the query to C because the responsibility of the corresponding hashline segment was previously delegated to this node. Finally, C provides the necessary route information to X.

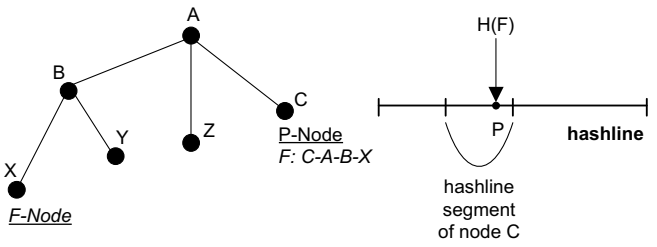

Fig. 1. An example WANET with six nodes.

The next section describes the details of the design along with the operations of the system.

4. Operations of the system

There are several basic operations supported by the system to locate files and determine the download route to enable file sharing. A *Node-Join* operation is carried out when a node is connected to a file sharing enabled WANET and a *Network-Join* is carried out when two file sharing enabled WANETs are merged. An *Access-P-Node* operation is used to find and access the node which stores the segment of the *hashline* including a desired point P. An *Access-F-Node* operation is used to find and access the node which stores a desired file with name F. *Insert* and *Delete* operations are used to add a file to the network (i.e. enable sharing) or remove a file from the network. A *Recover* operation is carried out to preserve the consistency between the actual location of shared files and the hash table storing the routing information when a disconnection with an adjacent node is detected. Finally, a *Leave* operation is carried out when a node decides to leave the file sharing enabled WANET.

Some of the operations mentioned above involve other operations (e.g. *Join* involves *Access-P-Node* and *Insert*). Detailed information about each operation is given in the following subsections.

4.1. Node-join

Whenever a node *N* decides to join a file sharing enabled WANET, the following steps are executed:

(1) *N* connects to an already existing node *K* of the network, which is accomplished by the underlying protocols specific to the WANET.
(2) *K* assigns a portion of its segment of the *hashline* to *N* and passes related hash table entries to it.
(3) *N* adds *K* to the routing path information maintained at each hash table entry for files indexed at *N* before saving the hash table entries.
(4) *N* assigns *K* as its parent and *K* adds *N* to its list of children in the logical tree-structure.
(5) *N* calls the *Insert* operation for each file it wants to share and whose hashed value is out of its responsibility.

As can be noticed, the *hashline* segment assigned to the new node is not randomly determined. Instead, the node to which the new node directly connects shares some portion of its responsibility on the *hashline* (e.g. half of it). This simple design is crucial for

easy and efficient routing of location queries to the nodes that can answer them. The corresponding operation is described in detail in Section 4.3.

4.2. Network-join

Let the nodes *N* and *K* be the members of two distinct file sharing enabled WANETs, *N*-Net and *K*-Net respectively, which are going to merge through an *N*–*K* connection. To obtain a larger file sharing enabled WANET from two smaller ones, the following steps are executed:

(1) *N* and *K* decide on which node is going to share its responsibility on the *hashline* or equivalently, which one is going to be the parent of the other. (In what follows, we will assume that *N* is chosen as the one to share its area of responsibility using some decision criteria).
(2) Every node of *K*-Net on the path from node *K* to the root of *K*-Net (the node with no parent), exchanges the parent–child role with its parent, including node *K* and the root of *K*-Net. That is every node on the specified path adds its former parent to its children list and it becomes the parent of its former parent. In this way, *K* becomes the new root of *K*-Net.
(3) *K* is connected to *N*, hence *N* becomes the parent of *K*.
(4) Based on the new parent–child relationships among *K*-Net nodes and *N*, starting from node *N* each parent shares some portion of its responsibility on the *hashline* with its children, in an iterative manner.
(5) Each node in *K*-Net calls the *Insert* operation for each file it wants to share.

4.3. Access-p-node

Whenever a node *N* wants to access *P-Node* (i.e. the node which is responsible for the segment of the *hashline* containing point P), it invokes the *Access-P-Node* operation. A node *K* receiving an *Access-P-Node* request follows these rules:

(1) If point P is included by the segment of the *hashline* that *K* is responsible for, *P-Node* is found and is *K*.
(2) If point P is included by the segment of the *hashline* that one of the children of *K* is responsible for, *K* adds itself to the route list and forwards the *Access-P-Node* request to the relevant child node.
(3) Otherwise: *K* adds itself to the route list and forwards *Access-P-Node* request to its parent.

Note that initially $N = K$, i.e. *N* applies the *Access-P-Node* operation to itself. Also note that the *P-Node* finally has the routing information between the node issuing *Access-P-Node* request (i.e. node *N*) and itself, since each node on the path from *N* to *P-Node* adds itself to the routing information carried inside the *Access-P-Node* request.

4.4. Access-f-node

Whenever a node *N* wants to access *F-Node* (i.e. the node which contains the file with name F), it invokes the *Access-F-Node* operation, which consists of the following steps:

(1) *N* hashes F and determines P, that is P = hash(F).
(2) Having point P, *N* invokes the *Access-P-Node* operation with *F-Node* location request, that is *N* asks *P-Node* the routing information from *P-Node* to *F-Node*.
(3) Having the route information back to *N*, due to the feature of *Access-P-Node*, the *P-Node* sends to *N* the route from itself to *F-Node* (remember that the route from *P-Node* to *F-Node* is stored as part of the hash table entry corresponding to point P).
(4) *N* combines the route information from itself to *P-Node* and from *P-Node* to *F-Node* and constructs the route necessary to access the *F-Node*.

4.5. Insert

Whenever a node *N* wants to share a file with name F, it invokes the *Insert* operation, which consists of the following steps:

(1) *N* hashes F and determines P, that is P = hash(F).
(2) Having point P, *N* invokes the *Access-P-Node* request with insertion as the request type and F as the filename.
(3) Upon receiving the request, the *P-Node* stores the filename F and the route information back to *N*, which is obtained during *Access-P-Node* operation, as part of the hash table entry created.

4.6. Delete

Whenever a node *N* wants to stop sharing a file with name F, it invokes the *Delete* operation, which consists of the following steps:

(1) *N* hashes F and determines P, that is P = hash(F).
(2) Having point P, *N* invokes the *Access-P-Node* operation with deletion as the request type and F as the filename.
(3) Upon receiving the request, the *P-Node* removes the entry for the file with name F from the hash table.

4.7. Recover

Whenever a node *N* determines a disconnection with one of its child nodes *K*:

(1) *N* regains the responsibility of the *hashline* segment that *K* was responsible for.
(2) *N* broadcasts to the WANET a message that includes information about the regained segment to force all the nodes to invoke *Insert* operation again for the files whose hashed names are included by the segment that *K* used to be responsible for. In this way, node *N* will have the hash table entries created for these files.

Whenever a node *K* determines a disconnection with its parent node *N*:

(1) *K* takes the full *hashline* as its area of responsibility.
(2) Starting from *K* each parent shares some portion of its responsibility on the *hashline* with its children.
(3) Each node calls the *Insert* operation for each file it wants to share.

4.8. Leave

When a node *N* wants to leave the file sharing enabled WANET, it invokes the *Leave* operation, which consists of the following steps:

(1) N invokes the *Delete* operation for each file it shares after which all index information about the files stored in N is removed from the WANET.
(2) N gives its responsibility for its segment of the *hashline* to its parent.
(3) N informs its parent P_N and children $C_1, C_2, \ldots, C_n$ about its departure to make sure P_N adds $C_1, C_2, \ldots, C_n$ to its children list and $C_1, C_2, \ldots, C_n$ assign P_N as their parent.

Note that the third step is possible only if all the children of node N are in the communication range of P_N. For the children that are not in the communication range of P_N, the *Recover* operation is executed.

Due to the nature of ad-hoc networks, nodes are not expected to notify the network upon leaving it. However, there may still be cases where the *Leave* operation is beneficial. Otherwise, the *Recover* operation handles the situation, but with a higher communication cost.

5. A sample scenario

After specifying each operation supported by the system, this part of the paper presents a sample scenario in which the system's operation can be observed. Suppose that initially two nodes called A and B meet. A includes files A1, A2, while B has B1, B2, B3. B discovers A, or in other words, B joins the network which is only composed of A. Previously, A was responsible of all the *hashline* and files A1 and A2 were mapped on to this line as depicted in Fig. 2(a). As explained in Section 4.1, when B is connected to A, A divides the entire *hashline* into two and gives one segment to B. Since A2 falls within the segment that B is now responsible for, A sends the location information (index information) for file A2 to B. The previous location information for A2 was null, meaning that the file was stored at the same node as where the location information is kept. But, from now on, B stores an index entry for A2 with location information like [A2, A]. Then B executes an *Insert* operation for files B1 and B2, since these are the files owned by B but not mapped to the part of the *hashline* that B is responsible for. Now, A stores location information [B1, B] and [B2, B] for these files as depicted in Fig. 3(b).

Suppose that a new node C discovers B and connects to it. Again a *Node-Join* operation will be invoked and the *hashline* segment that B is responsible for will be divided into two parts, as depicted in Fig. 2(c). C stores and shares files C1 and C2, which map to the points on the *hashline* as shown in the figure. First of all, B sends

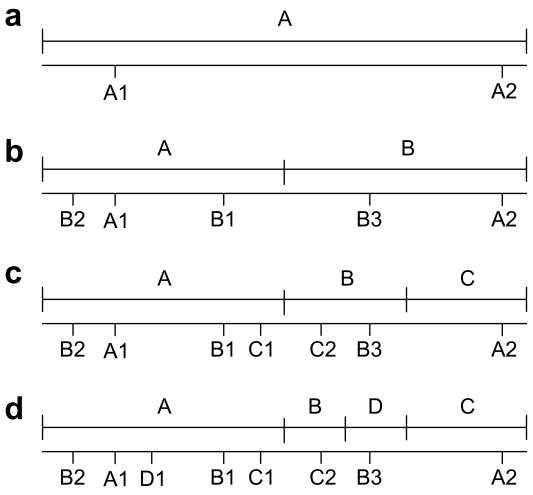

Fig. 2. Hashline states during network formation.

information about A2 to C, since A2 now falls in C's segment of responsibility. C should not only keep information about the node where file A2 can be found, but also keep path information about how can be reached from C. C also adds B to the path information and stores an index entry like [A2, BA]. This indicates that file A2 is stored at node A (right-most node in the path) and the path from C to that node is "AB". Next, C invokes the *Insert* operation both for C1 and C2. C1 maps to the segment controlled by A and C2 maps to the segment controlled by B. Therefore, an *Access-P-Node* request reaches B for file C2, and A for file C1. So, corresponding nodes store file names together with their route information to the node where files are actually stored. The route information is obtained during the path traversals of the *Access-P-Node* requests. The current state of location and routing information that is maintained in the network can be observed in Fig. 3(c). As the last member of the network, D, discovers B and connects to it. B, again divides the *hashline* segment it is responsible for into two parts and sends information about B3 to D. After that, D sends information about a single file it owns, D1 to A using an *Insert* operation. The final view of the *hashline* and the network topology together with distributed index information can be observed in Figs. 2(d) and 3(d), respectively.

Now, assume that D needs file A2. D does not know where the file A2 resides or even whether such a file exists or not. However, according to the hash value of the filename, it is known that this information is held by another node. D has only one neighbor, B (as its parent) to which the query is forwarded. So, B receives the query, expressed as [A2, D], meaning that file A2 is requested by D. B has two neighbors, A and C. According to the hash value of the filename and the current state of the *hashline*, B decides to forward the query to C. This is because B knows that one of its children, C in this case, is responsible for the segment of the *hashline* that includes the point that represents the hash value of the name of the requested file. Otherwise, B would forward the query to its parent, A. When the query is forwarded to C, it is not guaranteed that it will be answered by C. C may have had other nodes connect to it since connecting to B, so it may forward the query to one of its children again by determining within which segment the point lies. However, it does not matter to B whether C or one of its descendants answers the query. B only knows that query should be forwarded towards C in order to be resolved. For this particular case, C does not have any children and C holds the location information for A2. The path to source at which the query is initiated is also attached to the query. In this way, C receives a query [A2, BD], which means that node D requested file A2 and its request reached through node B. This path is used in order to send the query response (location information), [A2, BA], back to node D. C generates a query response message, [A2, BA], targeted to D and including the source route information "CBD" that gives the path to be followed. C passes the response to the next node on the path, which is B. Again by looking to the path information in the response message, B passes the message to the next node on the path, which is D, the originator of the query to locate file A2. D receives the query response message and the message includes the location information [A2, BA]. Now, D knows that the file A2 is located at node A and D also knows two paths: the path from D to C (the node which holds the location information) and the path from C to A (the node which stores the file). Node D concatenates those paths (D-B-C-B-A) and then eliminates the unnecessary loop B-C-B. The result is "D-B-A", the path from D to A. This is the path from query originator D to the node A that stores and shares the file A2. By means of this path, file A2 can now be directly reached and downloaded from A. These steps are depicted in Fig. 4(a) through (c).

As more nodes join the file sharing enabled network as explained in Section 4.1, the tree-structure become more involved.

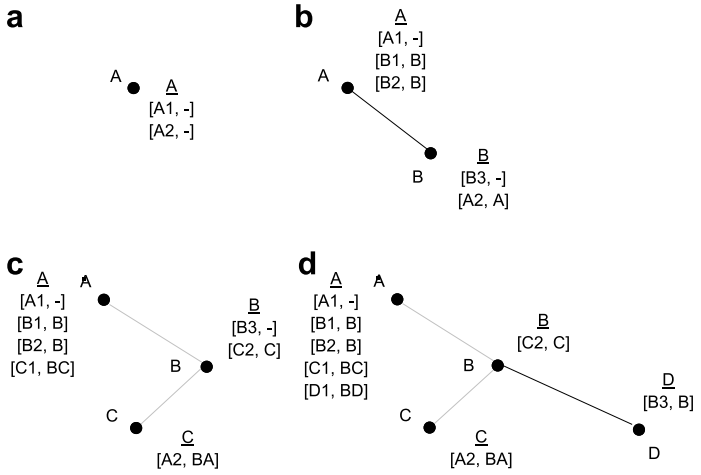

Fig. 3. Network topology and information distribution.

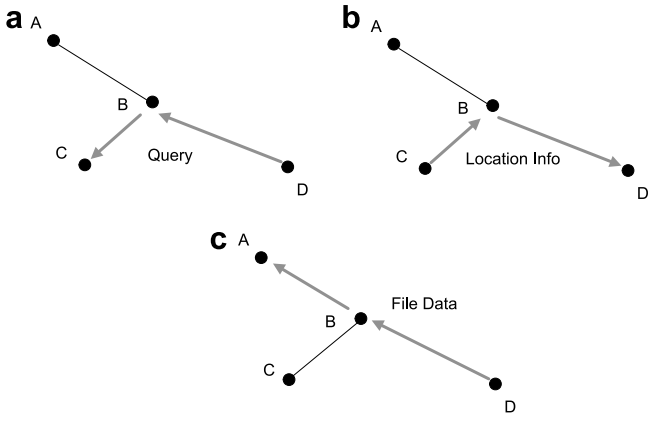

Fig. 4. File search and retrieval.

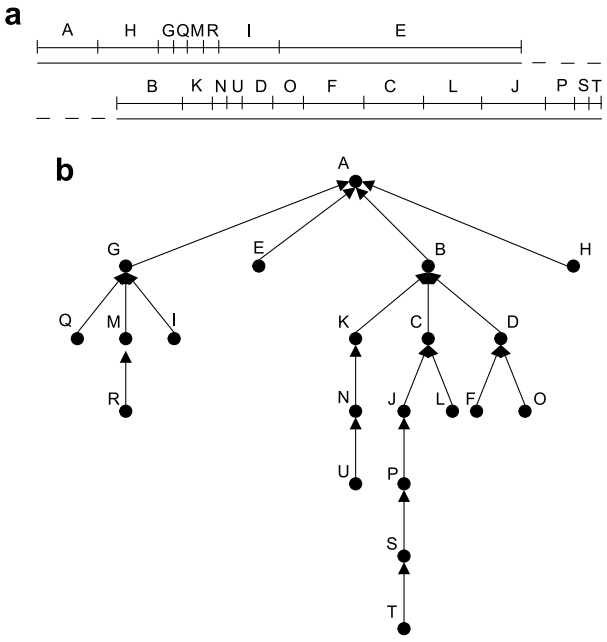

Fig. 5. Before network-join.

In Fig. 5, a later phase of the WANET shown in Figs. 2 and 3 is given. New nodes have joined to the WANET in alphabetical order. Connections between nodes can be inferred from the tree-structure given in Fig. 5(b). The state of the *hashline* is shown in Fig. 5(a).

Now, consider the case where the file sharing enabled WANET in Fig. 5 (WANET-1) merges with another file sharing enabled WANET shown in Fig. 6(a) (WANET-2) and assume that the connecting nodes are E of WANET-1 and V_5 of WANET-2. For such a merge operation, the *Network-Join* operation, which is explained in Section 4.2, is executed where nodes N and K in the operation correspond to the nodes E and V_5 in this sample scenario, respectively. In accordance with Step 2 of the *Network-Join* operation, all nodes on the path from node V_5 to the root node V_0, (i.e. V_5, V_2, V_0) exchange their parent–child relationships. The resulting parent–child relationships are depicted in the subtree, rooted at V_5, of the combined network shown in Fig. 6(b). Once the subtree rooted at V_5 is built, E shares a portion of its responsibility on the *hashline* with V_5. All descendants of V_5 share their responsibility on the *hashline* in a similar manner, iteratively. One possible distribution of responsibilities on the *hashline* among the nodes of the new combined tree is depicted in Fig. 6(c). Note that the resulting distribution may differ due to the order of children that a parent shares its responsibility with.

6. Simulation results

In order to measure the traffic overhead of the operations of our system on the network, a custom simulation environment [19], implemented in Java™, has been developed. [1] We define the traffic overhead for an operation to be the number of messages exchanged among the nodes of the network from initiation to the completion of the operation.

The simulation environment comprises an *Engine* that embodies the underlying network layers for the nodes of the WANET. It handles the intercommunication of nodes, informs about node discovery and notifies of link failures. On top of the *Engine*, multiple instances of a *Node* component operate according to the protocols of the system. Every *Node* instance runs in a separate thread, and they communicate with each other by message passing. All messages go through the *Engine*, which controls the ranges of the nodes and counts the number of messages exchanged.

Packet collisions and any possible effect of the underlying air protocol are not simulated and reflected in the simulation environment. Hence we assume no packet collisions and no interference in

[1] Java 2 SDK, Standard Edition (version 1.4.1) is used as the development environment.

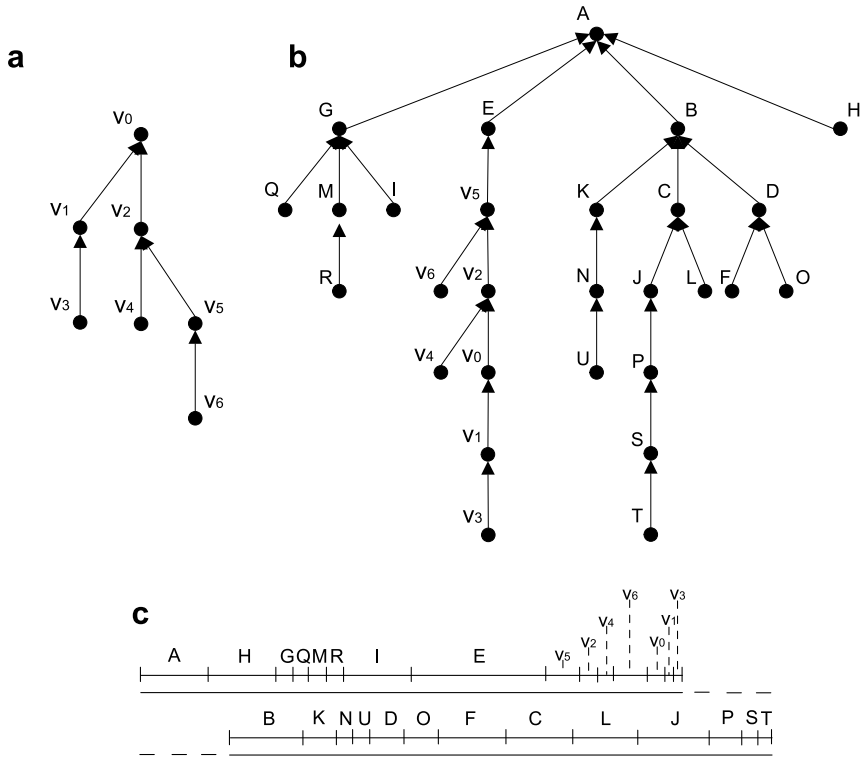

Fig. 6. Network-join.

our experiments. The dimensions of the rectangular network area and the communication range of nodes can be adjusted. In addition, the number of nodes, their locations (randomly assigned) in the network area, and the number and type of operations to be executed can be varied in each simulation execution.

6.1. Traffic overhead of system operations

For the measurement of traffic overhead, our simulation environment generated 100 random topologies. An operation was executed on each of these topologies 100 times. In each execution, related nodes and/or values were selected randomly. In order to measure the traffic overhead of an *Insert* operation, for instance, a randomly selected node was made to initiate the *Insert* operation regarding a randomly selected P value, which in reality would be the hash value of a file name. In total, 10,000 experiments were performed for each operation and the number of messages exchanged was counted during each experiment. Their arithmetic mean was computed as the average traffic overhead. This computation was performed for different number of nodes ranging from 10 up to 100 nodes in increments of 10. The results for each operation are depicted in Fig. 7. Note that the average traffic overhead of an *Access-P-Node* operation is equal to that of an *Insert* operation.

Since our system makes use of a tree shaped overlay network, the average traffic overhead of the *Insert* operation is proportional to $log_m n$, where n is the total number of nodes and m is the node degree. However, m differs for each generated topology and even for each node of a topology. Formation of the network does not guarantee a balanced tree-structure. That is why, we observe a linear increase in Fig. 7(a) but a less steep curve compared to the worst case (i.e. traffic overhead is equal to the number of nodes).

When the *Access-P-Node* request is initiated with *Access-F-Node* as the request type, the *P-Node* is reached first. From the *P-Node*, location information is returned to the source of the query. Having location information, the source of the query accesses the *F-Node* (Recall Fig. 4). On the other hand, when the *Access-P-Node* request is initiated with *Insert* as the request type, the *P-Node* is reached and location information recorded in the request is stored at the destination. No extra messages are exchanged. This can be verified with the simulation results in which the number of messages exchanged for the completion of the *Access-F-Node* operation is approximately 3 times the number of messages exchanged for the completion of the *Insert* operation (see Fig. 8).

As shown in Fig. 7(c), we also observe a linear increase in the overhead of the *Recover* operation. However, it is much higher compared to the results regarding the *Insert* and *Access-F-Node* operations. This is expected since, during the execution of the *Recover* operation, the *hashline* is redistributed in the whole sub-network, where the disconnected child node becomes the root.

Apparently, the *Network-Join* operation has the most traffic overhead among all operations, which is even much higher than the overhead of the *Recover* operation, as depicted in Fig. 7(d). Moreover, the average traffic overhead appears to follow asymptotically $O(n^2)$ with increasing number of nodes. The excessive number of *Insert* operations executed would cause this high cost. Recall that each node in the joining network must send *Insert* requests concerning the whole network for each file it shares (Each node shares x files, where x is a random integer between 1 and 10). Other than this, there is an overhead for the redistribution of the *hashline* as well but that was also an issue for the *Recover* operation and it did not lead to as much increase in the average traffic overhead.

6.2. Comparison with the flooding approach

The results obtained regarding the traffic overhead measurements are also used for comparing the performance of the proposed system with a WANET in which peer-to-peer file sharing is possible and the queries are simply flooded. Since the solution proposed in the paper virtually guarantees to find and access a

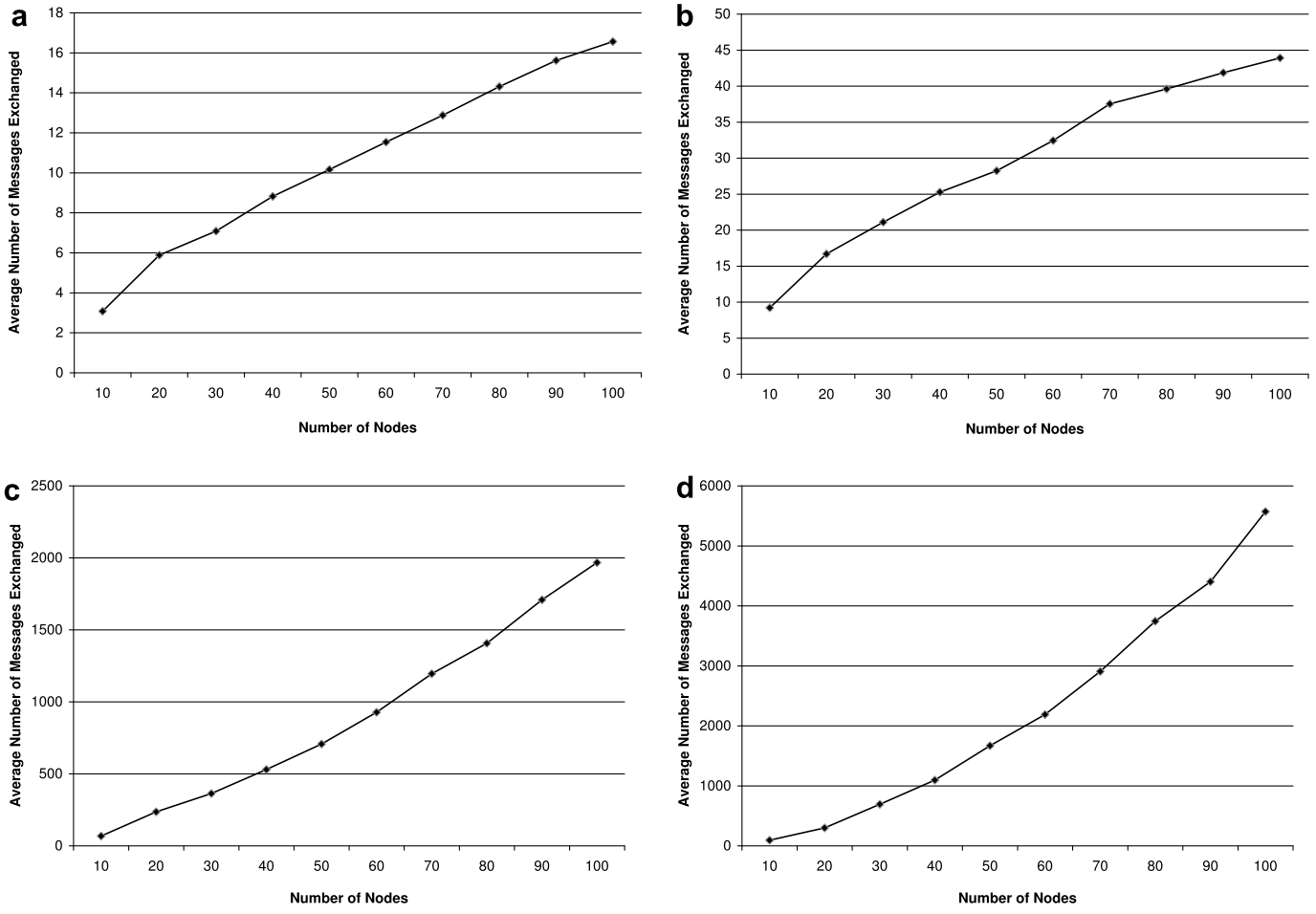

Fig. 7. Average traffic overhead measures for each operation.

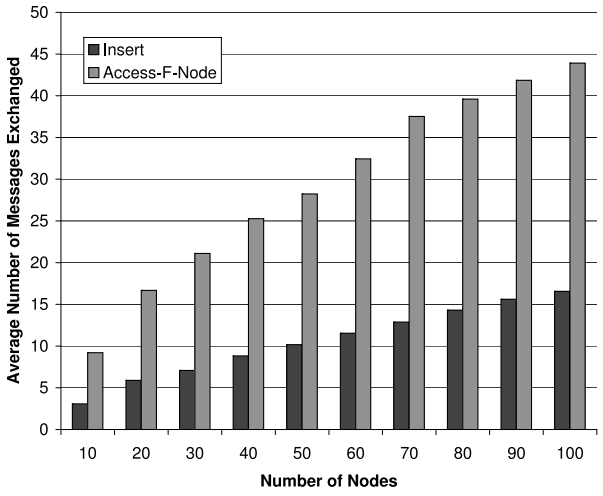

Fig. 8. The average traffic overhead comparison of the *Insert* and *Access-F-Node* operations.

file in the network if it is shared, the flooding model is assumed to have no facilities such as selective forwarding, which may prevent finding some of the shared files. Hence, it is assumed that whenever a query is initiated in the flooding-based network, all the nodes receive the query. The cost of such a system, f, is formulated with (1), where x is the number queries and n is the number of nodes:

$$f = x \times (n - 1) \tag{1}$$

Once the network is established, it is obvious that a single query is very cost effective in the proposed system compared to the flooding-based one. But, the *Recover* and the *Network-Join* operations cause an overhead each time a disconnection–reconnection cycle occurs, whereas a flooding-based system does not have such an overhead. Hence, the main objective of the comparison is to find out the number of queries that must be initiated to amortize the cost due to a single disconnection–reconnection cycle of the proposed system. It is expected that the proposed system would be more cost effective, as far as the number of messages exchanged is considered, after a certain number of queries initiated without *Recover* or *Network-Join* operations are executed. The cost model of the proposed system is given in (2). In the formula, x denotes the number of queries and d denotes the number of disconnection–reconnection cycles. The average costs of *Access-P-Node*, *Recover* and *Network-Join* operations are obtained from the simulation results:

$$c = x \times cost(Access - P - Node) + d \times (cost(Recover) + cost(Network - Join)) \tag{2}$$

In order to find the number of queries (i.e. x) required to amortize the cost of a disconnection-reconnection cycle in the proposed system, the d value is taken to be 1, f and c values are equated and x values are calculated for different network sizes. The results show that the total number of queries increase linearly from around 30 to around 90, as the number of nodes change from 10 to 100, which is also depicted in Fig. 9. Whenever the per-node query counts are

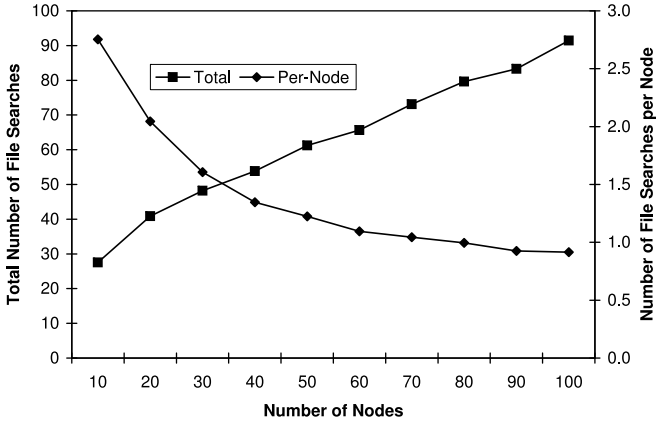

Fig. 9. Comparison of the proposed and flooding-based systems.

considered, as 3 queries in a 10-node network or less than 1 queries in a 100-node network are executed on the average, before a disconnection occurs, the proposed system becomes more bandwidth-efficient than a flooding-based network of the same size.

As the simulation results indicate, our system efficiently carries out file searches and it is also scalable. However, in the case of frequent disconnections, the system would introduce too much traffic overhead on the network. Consequently, we can state that our system works better on a WANET with a low rate of mobility and frequent file searches. If members of the file sharing enabled WANET continuously move around and search for files infrequently, then flooding would be a better approach to apply. In our system, we try to preserve consistent and distributed location information, in order to prevent flooding and access files through unicast queries. The cost we pay for keeping the location information consistent would be amortized by the exceedingly reduced cost of subsequent file searches. When such information does not exist, each file search performed by each node of the network would flood the whole network and lead to a congestion. Members of the file sharing enabled WANET can be mobile, but we presume that they perform file searches more than they move around. We investigate the effect of mobility in more detail in the following section.

6.3. The effect of mobility

To be able to measure the impact of mobility, we have implemented the commonly used *Random Waypoint Model* (RWM) [20]. We have tested the number of disconnections and the number of messages exchanged for different node velocities (V_{max} parameter in the RWM). We have set the T_{pause} parameter to 0, which means that the mobile nodes are constantly moving. We have generated 100 random topologies, each composed of 100 nodes, in an area of 1000 m$\times$ 1000 m. We have set the node range to be 100 m. We have set all the nodes of the network to be mobile and run the simulation for 1000 s. We have repeated the simulation for each random topology and calculated the average number of disconnections and the traffic overhead of recovery operations accordingly. The results can be seen in Fig. 10.

In Fig. 10, the *x*-axis shows the varying node velocities (i.e. V_{max}) and the *y*-axis shows the amount of messages that are exchanged to recover from the disconnections due to mobility. From the results, we can see that the number of messages exchanged for recovery operations asymptotically follows $O(n^2)$ with respect to the node velocities.

We have also calculated the number of file searches that is necessary to amortize the cost of mobility. For this, we have used the cost model in (2). We have performed measurements for varying

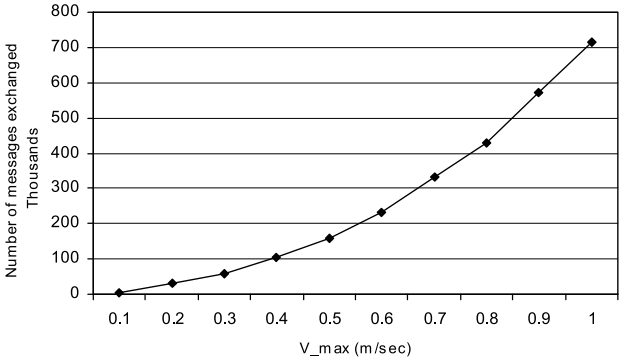

Fig. 10. The traffic overhead caused by recovery operations as a result of mobility.

node velocities and for different percentages of nodes that are set to be mobile in the network. The results can be seen in Fig. 11.

In Fig. 11, the *x*-axis shows the varying node velocities (i.e. V_{max}) and the *y*-axis shows the amount of queries per node that are necessary to amortize the recovery cost imposed by mobility. The three plots stand for networks where (a) all the nodes in the network are mobile, (b) 50% of the nodes in the network are mobile and (c) 10% of the nodes in the network are mobile. We can see from the results that the number of file searches that is necessary to amortize the cost of mobility increases linearly. This means that rehashing and keeping the network structure intact pays off as long as the nodes increase the number of their file queries as their mobility increases. Otherwise, in the case of high mobility and very infrequent file searches, flooding would be a better approach.

7. Conclusion and future work

In this paper, we proposed a peer-to-peer system that enables file sharing in wireless ad-hoc networks. The novel approach introduced in this work is the unification of lookup and routing functionality, which results in a cross-layer scheme. The system keeps track of the routing information together with the location information, which is fully distributed. In achieving this, we adapt techniques from peer-to-peer systems developed for wire-line networks as well as source routing techniques.

Simulation results showed that our system enables efficient access to shared files. However, it may not work efficiently when frequent disconnections occur. Nonetheless, we believe that the environment in which a file sharing system would be used is a WANET where mobility should be supported, but where the rate of mobility is not high. A conference room may be given as an example to such an environment, in which attendees need to share files.

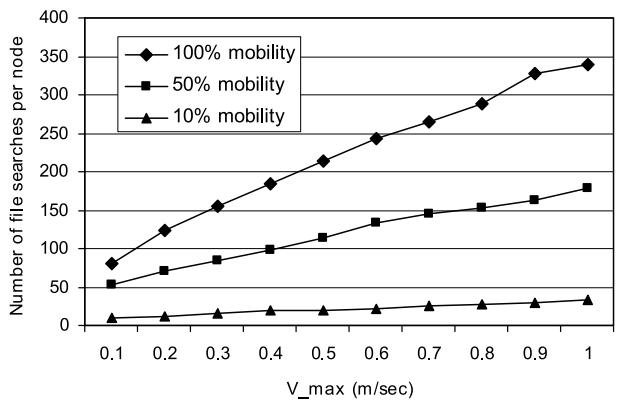

Fig. 11. The number of queries per node that is necessary to amortize the cost of mobility.

Although the mobility rate is not high, WANET should support the mobility and ad-hoc features, where there is no infrastructure support and the network is build up upon demand for a relatively short duration of time.

As a future work, the *Recover* and *Network-Join* operations can be modified, so that they have less traffic overhead. In any case, nodes can wait for some time when disconnections occur, hoping that the connection will be retained. In this way, intermittent disconnections for short time periods would not lead to frequent execution of *Recover* and *Network-Join* operations, one after the other.

A supplementary modification is possible in the hash function that is used to map file names to certain keys. In our system, we propose the usage of any uniform hash function for that purpose. This approach has a disadvantage in that only search of exact file names is possible. Instead of a uniform hash function, other mapping techniques can be used like Soundex [21], which maps similar names to same keys. In that case, a set of results will be returned as an answer to queries.

The responsibility sharing policy can also be reconsidered. Currently, in our specifications, the *hashline* segment is divided into two halves when it is shared. As an alternative, the dissection point can be determined according to the distribution of files, although this may change in time. Such an approach can especially be useful when a uniform hash function is not used.

Acknowledgement

We would like to express our thanks to Burcu Kaplanlıoğlu, Burcu Ayşen Ürgen and Selen Pehlivan for the implementation of the simulation environment.

References

[1] Napster protocol specification, 2007, [Online], Available: http://opennap.sourceforge.net/.
[2] Gnutella A Protocol for a Revolution, 2007, [Online], Available: http://rfc-gnutella.sourceforge.net.
[3] The FastTrack project, 2007, [Online], Available: http://developer.berlios.de/projects/gift-fasttrack.
[4] Bluetooth, Bluetooth Special Interest Group, [Online], Available: http://www.bluetooth.com.
[5] D. Johnson, D. Maltz, Dynamic source routing in ad-hoc wireless networks, in: Proceedings of SIGCOMM96, ACM, California, USA, 1996.
[6] E. Royer, C.-K. Toh, A review of current routing protocols for ad hoc mobile wireless networks, IEEE Personal Communications (1999).
[7] C. Perkins, E. Belding-Royer, S. Das, Ad hoc On-demand Distance Vector (AODV) routing, July 2003, RFC 3561.
[8] M. Gerla, C. Lindemann, and A. Rowstron, P2P manets - new research issues, in: Perspectives Workshop: Peer-to-Peer Mobile Ad Hoc Networks – New Research Issues, Dagstuhl, Germany, 2005.
[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: Proceedings of SIGCOMM01, ACM, California, USA: ACM, 2001.
[10] S. Androutsellis-Theotokis, D. Spinellis, A survey of peer-to-peer content distribution technologies, ACM Computing Surveys (CSUR) 36 (4) (2004) 335–371.
[11] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for internet applications, in: Proceedings of the ACM SIGCOMM 01 Conference, San Diego, California, USA, August 2001.
[12] M. Papadopouli, H. Schulzrinne, Effects of power conservation wireless coverage and cooperation on data dissemination among mobile devices, in: Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2001), ACM, Long Beach, California, USA, 2001.
[13] C. Lindemann, O. Waldhorst, A distributed search service for peer-to-peer file sharing in mobile applications, in: Proceedings of the Second IEEE Conference on Peer-to-Peer Computing, Linkping, Sweden, September 2002, 7381 pp.
[14] A. Klemm, C. Lindemann, O.P. Waldhorst, A special-purpose peer-to-peer file sharing system for mobile ad hoc networks, in: Proceedings of the Vehicular Technology Conference, 2003, vol. 4, October 2003, 27582763 pp.
[15] A. Duran, C. Shen, Mobile ad hoc P2P file sharing, in: Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2004), Orlando, FL, 2004.
[16] R. Gold, C. Mascolo, Use of context-awareness in mobile peer-to-peer networks, in: Proceedings of the Eighth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS01), Bologna, Italy, 2001.
[17] T. Camp, J. Boleng, L. Wilcox, Location information services in mobile ad hoc networks, in: Proceedings of the IEEE International Conference on Communications (ICC 2001), 2001.
[18] M. Caesar, M. Castro, E.B. Nightingale, G. OShea, A. Rowstron, Virtual ring routing: network routing inspired by DHTs, SIGCOMM Computer Communications Review 36 (4) (2006) 351362.
[19] H. Sozer, A peer-to-peer file sharing system for wireless ad-hoc networks, Masters thesis, Department of Computer Engineering, Bilkent University, 2004, [Online], Available: http://www.thesis.bilkent.edu.tr/0002646.pdf.
[20] J. Broch, D.A. Maltz, D.B. Johnson, Y.-C. Hu, J. Jetcheva, A performance comparison of multi-hop wireless ad hoc network routing protocols, in: Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom98), ACM, 1998.
[21] D. Knuth, The Art of Computer Programming, Second ed., vol. 3, Addison-Wesley, 1998. Sorting and Searching.