A Path-Quality-Aware Peer-to-Peer File Sharing Protocol for Mobile Ad-hoc Networks: Wi-Share

Efe Karasabun*, Doğuş Ertemür†, Seyhun Sarıyıldız†, Metin Tekkalmaz* and Ibrahim Korpeoglu*

*†Department of Computer Engineering, Bilkent University, Ankara, Turkey
Email: *{efe, metint, korpe}@cs.bilkent.edu.tr, †{dertemur, seyhuns}@alumni.bilkent.edu.tr

***Abstract*—Peer-to-peer networks are rather well-studied and currently there are numerous systems based on peer-to-peer principles running on the Internet. On the other hand peer-to-peer networks for mobile ad-hoc networks have attracted attention only in the recent years. In this paper, we propose a novel peer-to-peer file sharing system particularly designed for mobile ad-hoc networks. The proposed system, namely Wi-Share, has both network and application layer aspects enabling efficient search and download of the shared files. Wi-Share uses reactive routing for the search operation combined with source discovery and uses the routing tables constructed during the search operation for the download operation. In order to increase the overall efficiency of the file sharing in the network, Wi-Share applies techniques to reduce the required traffic and to increase efficient parallelism of the download operation. These techniques include filtering search results, preferring the higher quality routing paths, using partitioned download scheme and allowing the nodes that have joined to the network recently to contribute to the ongoing downloads. Wi-Share is implemented to work on mobile computers and the results of several experiments are also presented in the paper.**

I. Introduction

Peer-to-peer (P2P) file sharing applications designed and implemented for the Internet has been very successful and popular throughout Internet users since the mid-90s. Ad-hoc networks, which provide decentralized, mobile and infrastructureless communication, have also been evolving since the mid-90s by the standardization and increased availability of the wireless adapters (e.g. 802.11). The system proposed in this paper, naming Wi-Share, aims to provide an efficient solution for P2P file sharing in mobile ad-hoc networks, which involves specialized routing and application layer protocols.

At the network layer, the routing protocol proposed in this paper aims to maintain the overall throughput as high as possible for the whole network by finding optimum paths while trying to minimize the use of power resources. At the application layer, on the other hand, mechanism such as filtering and parallel downloads, decrease the bandwidth usage and increase download efficiency. Hence, an efficient medium is provided for the fundamental P2P file sharing operations, which are search and download.

II. Related Work

As far as P2P file sharing is considered, Napster [1] appears to be one of the earliest and most popular applications. 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. Napster enables easy location lookup by using a central server, but it is affected by the typical weaknesses of centralized systems.

Napster is designed for the Internet and it is not suitable for mobile ad-hoc networks (MANETs) due to the dynamic nature of such networks and lack of infrastructure. A recent work [9] analyze and test the performance of Gnutella [2] on a MANET using AODV and OLRS as the underlying MANET routing protocols. They conclude that Gnutella working on top of MANET routing protocols is not suitable for MANETs because of the unnecessary network overhead generated by Gnutella and because Gnutella is not suitable to handle cases specific to MANETs such as node mobility and network partitioning. Therefore they suggest a cross-layer optimization for the Gnutella protocol. This cross-layer optimization involves integrating Gnutella specific messages to the periodic messages exchanged by the OLRS protocol to achieve better peer discovery and route selection leading to less network overhead and more resistance to node mobility and network partitioning. However, the main shortcoming of this approach is that it does not consider the residual battery power levels and the data traffic generated in the network when selecting routes.

ORION, described in [5] and as a P2P file sharing approach for MANETs, 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. The main difference of Wi-Share from ORION is that Wi-Share considers the quality of the paths being used for the download operation. Wi-Share's approach allows better network utilization by considering the power levels of the nodes and amount of data transfer being performed. Consideration of power level and amount of data being relayed is an important feature in battery constrained mobile ad hoc networks. Another P2P file sharing system for wireless ad-hoc networks based on distributed hash tables is proposed in [10]. The same study also includes experimental results comparing the proposed system and the flooding-based systems. As presented in [10], flooding based solutions for P2P file sharing in mobile ad-hoc networks are superior to the distributed hash table based solutions as long as the mobility in the network is high.

III. Wi-Share P2P File Sharing Protocol

A. Search Operation

The search operation consists of making a search with one or more keywords in the P2P file sharing system and obtaining the corresponding search results. Wi-Share uses a reactive (i.e. on-demand) routing approach for the search operation. The search operation involves the route discovery phase in which search requests are flooded through the network as the user initiates the search. The routing protocol used in Wi-Share has similarities with both ad-hoc on-demand distance vector routing (AODV) [4] and dynamic source routing (DSR) [3]. How these approaches combined while propagating the search request packets and search result packets through the network is explained in the subsequent sections.

1) Search Request Packet Propagation: The search request packet is composed of the unique search request broadcast id that is constructed using the current system time in milliseconds plus some random number, search keywords, the path indicating the nodes that the search request packet has traversed so far and an "availability parameter" set by each of those nodes. The availability parameter of a node consists of its current available power, its path distance to the node requesting the search, and the amount of total data it has sent, received or forwarded recently. The availability parameter is used to find out the quality of a path, as explained in Section III-B, and consequently to make routing decisions, since it reflects the status of the node in terms of how much it may contribute to the new downloads.

When a node receives a search request packet, there are three operations that it should perform. First is to update its routing table, which stores the "next hop" information for various destinations, by using the path information obtained from the search request packet. This is called reverse path establishment and explained in Section III-A2. Second, if the receiver node has matching files with the search keywords, it prepares a search result packet and sends it to the search request packet initiator as explained in Section III-A3. Third, each receiver node of the search request packet broadcasts the search request packet to its neighbors. Before broadcasting the packet, the receiver node adds itself and its availability parameter to the path that the search request packet goes through. This way the quality of the path on which the search request packet flows can be identified and analyzed.

Algorithm 1 shows the steps that a node follows when it receives a search request packet. In the algorithm, id is the unique identifier of the search request, kw is the keywords currently being searched for, $path$ is the path that the search request packet has traveled so far, and $availability$ is the availability parameters of the nodes on the $path$. Furthermore, $cache$ is the search request cache, $cache[id]$ is a cache entry corresponding to the search request with identifier id, and $\sum availability$ represents the path quality computed using $availability$. Therefore, $cache$ stores path quality values for each search request received by the node.

A node discards a search request packet if the node has broadcast due to the same request before and the current receive path is not better than the one caused the previous broadcast. This approach is applied in order to reduce the overhead of the search broadcasts while still promoting the most available routes. Nodes recognize previous broadcasts using a search request cache. Entries of the search request cache are deleted after a timeout period. See Figure 1 for a sample search request propagation scenario.

Algorithm 1 Search request packet propagation

```
 1: func ProcessSearchRequest($id$, $kw$, $path$, $availability$)
 2: if $path$ is new or $\sum availability$ of $path$ is better then
 3:     update routing table
 4: end if
 5: if ($cache[id] \notin cache$) or
        ($\sum availability$ is better than $cache[id]$) then
 6:     $cache \leftarrow cache \cup cache[id]$
 7:     $cache[id] \leftarrow \sum availability$
 8:     if $\exists$ a file with $kw$ then
 9:         send search result packet
10:     end if
11:     broadcast search request packet
12: end if
```

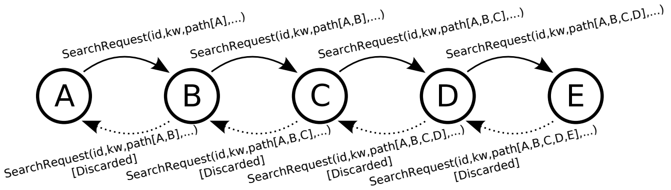
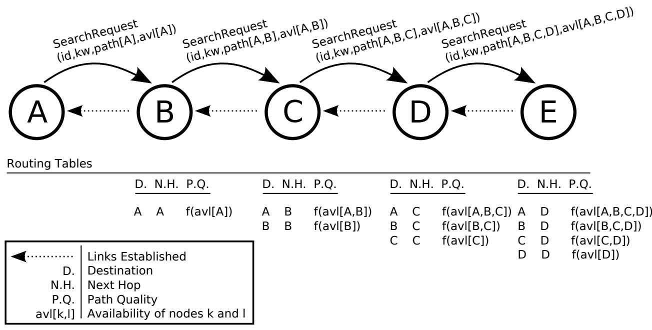

SearchRequest(id,kw,path[A],...)
SearchRequest(id,kw,path[A,B],...)
SearchRequest(id,kw,path[A,B,C],...)
SearchRequest(id,kw,path[A,B,C,D],...)
A
B
C
D
E
SearchRequest(id,kw,path[A,B],...) [Discarded]
SearchRequest(id,kw,path[A,B,C],...) [Discarded]
SearchRequest(id,kw,path[A,B,C,D],...) [Discarded]
SearchRequest(id,kw,path[A,B,C,D,E],...) [Discarded]

Fig. 1. Our search algorithm is based on classic flooding operation, however some precautions are taken to minimize negative effects of flooding and save resources of the network. Search request packets are broadcast only if not done so before or the current receive path is better than the path caused the previous broadcast.

The search request packets are processed at the application layer of Wi-Share, since the keywords can be processed and the availability parameter can be obtained at this layer of the protocol stack. It is important to note here that although the packets used in search operation goes up to the application layer, the download operation takes place at the network layer of the protocol stack making the download operation relatively fast as explained in Section III-C.

2) Reverse Path Establishment: When a node receives a search request packet, it establishes a reverse link to the node which has initiated the search operation. The quality of the path between the current node and the node that has initiated the search is computed from the availability parameters of each node on the path, which are obtained from the search request packet. The reverse link establishment is done by setting the "next hop" information of the node initiating the search request as shown in the sample scenario depicted in Figure 2. Reverse path establishment is shown between lines 2 and 4 of Algorithm 1.


SearchRequest (id,kw,path[A],avl[A])
SearchRequest (id,kw,path[A,B],avl[A,B])
SearchRequest (id,kw,path[A,B,C],avl[A,B,C])
SearchRequest (id,kw,path[A,B,C,D],avl[A,B,C,D])
A
B
C
D
E
Routing Tables
Links Established
D. Destination
N.H. Next Hop
P.Q. Path Quality
avl[k,l] Availability of nodes k and l

D.	N.H.	P.Q.	D.	N.H.	P.Q.	D.	N.H.	P.Q.	D.	N.H.	P.Q.
A	A	f(avl[A])	A	B	f(avl[A,B])	A	C	f(avl[A,B,C])	A	D	f(avl[A,B,C,D])
			B	B	f(avl[B])	B	C	f(avl[B,C])	B	D	f(avl[B,C,D])
						C	C	f(avl[C])	C	D	f(avl[C,D])
									D	D	f(avl[D])

Fig. 2. A sample reverse path establishment scenario. The figure presents how the nodes update their routing tables as the search request packet travels through the network. Each node extracts the path information of the packet and finds possible routes to different destinations. Those routes are compared with current routing table entries. If a new route is found for a specific destination, an entry is created. If a better route, in terms of the path quality, is found for an existing destination, current entry of the destination is replaced with an entry describing the better route.

It is important to note that when a search request packet comes to any node, that node extracts the path information and obtains the availability parameters towards the nodes included in the packet. If the obtained availability parameters towards some destination is better than the prior availability parameters that are known towards that destination, than the new availability parameters are used to decide on the new next hop towards that destination. How the availability parameters are used to select paths and route packets is explained in detail in Section III-B. This approach allows the intermediate nodes to use fresher and better paths when relaying packets for ongoing download operations.

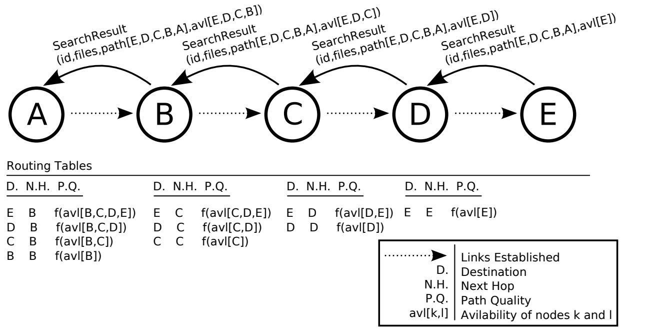
3) Search Result Packet Propagation: If a node has files that correspond to a received search request packet, it creates a search result packet and sends the search result packet by using the reverse of the path indicated inside the search request packet. Such a reverse path is assumed to lead to the node which has initiated the search operation, since Wi-Share assumes symmetric links between the nodes. The search result packet is composed of search request identifier, list of the files that corresponds to the search keyword, the path of the search request packet and the availability parameters of each node in the path. The search result packet is propagated back to the search owner as explained in Algorithm 2. In the Algorithm 2, different from Algorithm 1, $path$ contains all the nodes between the source (i.e. node containing the files matching the keywords) and the destination (i.e. node that initiated the search). Additionally, $file$ is the list of files matching the keywords in the corresponding search request. Nodes receiving search result packet applies filtering, as explained in Section III-A5, if necessary.

4) Forward Path Establishment: While the search result packet is sent back to the node which has initiated the search request, forward links between the nodes on the path and the node which has initiated the search result packet is established. The forward path establishment is similar to the reverse path

Algorithm 2 Search result packet propagation

```
 1: func ProcessSearchResult($id$, $file$, $path$, $availability$)
 2: if $path$ is new or $\sum availability$ of $path$ is better then
 3:     update routing table
 4: end if
 5: $target \leftarrow$ first element of $path$
 6: if $target \neq current\text{-}node$ then
 7:     filter $file$
 8:     if $file \neq \emptyset$ then
 9:         forward search result packet to the next hop in $path$
10:     end if
11: end if
```

establishment with the difference that forward path establishment is done by setting the routing table entries at the nodes relaying the search result packet destined to the search result source. If the node already has a routing table entry destined to the search result source, then the node decides whether to update routing information by comparing the quality of the previous and the current paths. If the quality of the new path is higher than the one in the routing table, then the routing table entry is updated accordingly as explained in the sample scenario depicted in Figure 3. Lines between 2 and 4 corresponds to forward path establishment in Algorithm 2.

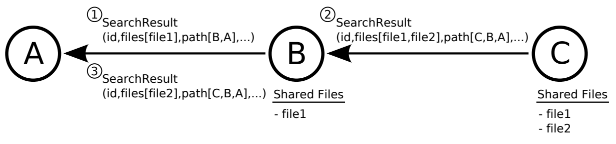

SearchResult (id,files,path[E,D,C,B,A],avl[E,D,C,B])
SearchResult (id,files,path[E,D,C,B,A],avl[E,D,C])
SearchResult (id,files,path[E,D,C,B,A],avl[E,D])
SearchResult (id,files,path[E,D,C,B,A],avl[E])
A
B
C
D
E
Routing Tables
Links Established
D. Destination
N.H. Next Hop
P.Q. Path Quality
avl[k,l] Avilability of nodes k and l

D.	N.H.	P.Q.	D.	N.H.	P.Q.	D.	N.H.	P.Q.	D.	N.H.	P.Q.
E	B	f(avl[B,C,D,E])	E	C	f(avl[C,D,E])	E	D	f(avl[D,E])	E	E	f(avl[E])
D	B	f(avl[B,C,D])	D	C	f(avl[C,D])	D	D	f(avl[D])			
C	B	f(avl[B,C])	C	C	f(avl[C])						
B	B	f(avl[B])									

Fig. 3. A sample forward path establishment scenario. Node E, the search result packet source, prepares the packet by filling matching files, full path of the result packet, which is the reverse of corresponding search request packet's path, and its own availability parameter. Then it sends the packet to the next node according the path, which is node D. Each node on the path appends its own availability parameter, filters the matching files and passes the packet to the next node. They also maintain the route information about the network as they process the search result packets. As an example, node B learns paths to nodes C, D and E with a single packet.

5) Filtering: While the search result packets are sent to the node which has initiated the search operation, in order not to waste resources and bandwidth, intermediate nodes are required to check whether they already have the files contained in the received search result packet. If any of those files is available in the relaying node, the node removes those entries from the search result packet and forwards the remaining ones. This operation is called filtering. By the use of filtering, the files that the intermediate nodes have will travel less and therefore the intermediate nodes are no more required to make unnecessary routing. This idea is borrowed from the

ORION Search Algorithm [5]. (See Figure 4). Forward path establishment is shown in line 9 of Algorithm 2.



①SearchResult (id,files[file1],path[B,A],...)
②SearchResult (id,files[file1,file2],path[C,B,A],...)
③SearchResult (id,files[file2],path[C,B,A],...)
A
B
C
Shared Files
- file1
Shared Files
- file1
- file2

Fig. 4. A sample filtering scenario. Assume that node A issues a search for which file1 and file2 are valid responses. First, node B responds the search request of node A with file1. Next, node C responds with two files, file1 and file2 through node B. Since it is not reasonable to download file1 from both nodes B and C, node B filters the search result packet of node C before forwarding it, in order to hide the unnecessary results.

B. Path Quality, Path Cost Computation and Availability Parameters

The use of availability parameter introduced by Wi-Share is a novel approach for making better routing decisions in terms of the efficiency of the whole ad-hoc network. Availability parameters of the nodes on a path are used to compute the quality of that path, which, in turn, is used as a basis for routing decisions. Therefore, routing decisions are based on three parameters; the length of the paths, the available (residual) power levels on the paths and traffic load on the paths. Path quality combines these parameters to achieve fairness and efficiency for all nodes.

The quality of a path is inverse of its cost which is computed as in (1), where PC is the path cost, $HopCount$ is the number of hops between the nodes, $Battery_{min}$ is the battery level of the node with the least battery power on the path, and $Traffic_{max}$ is the traffic load of the node with the highest load. C_h, C_b, and C_t are hop count, battery level and traffic load constants respectively.

$$\begin{aligned} PC &= C_h \times HopCount \\ &+ C_b \times (100 - Battery_{min}) \\ &+ C_t \times Traffic_{max} \end{aligned} \qquad (1)$$

The default values of the constants are selected as to adjust the PC value in a way to respond the possible ranges of hop count, battery level and traffic load. Generally we have observed that the hop count is in the range [0-10], battery level is in the range [0-100] and traffic load (measured in packets per second) is in the range [0-600].

The reason why Wi-Share uses the above mentioned parameters for routing decisions is because these parameters are observable at each node without incurring any overhead to the protocol. Link quality metrics such as ETX [6] (expected transmission count), which is the expected number of transmissions a node requires to successfully transmit a packets to a neighbour, ML [7] (minimum loss), which is based on finding the routes with the lowest end-to-end loss probability and ENT [8] (effective number of transmissions), which measures the number of successive retransmissions per link considering the variance, requires that peridic link quality control packets are broadcast to neighbours. Broadcasting of periodic control packets would have caused extra network overhead at Wi-Share protocol which works on an reactive basis rather than a pro-active basis.

Figure 5 shows how the availability parameter, and consequently the path costs, allows to select a better path avoiding congestion on a single node and using up battery from a single node.

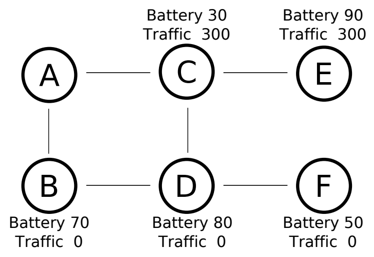


Fig. 5. In the network, there is an ongoing data transfer between nodes C and E. When the node A searches the network for a file that is only shared by nodes C and F, it receives replies from up to 3 different paths. If the search request packet following path C-A arrives to node C before the search request packet following path C-D-B-A, then search result packets are sent back from both paths. Otherwise the search result packet is sent only to follow path C-A, since the PC value of this path is computed to be lower on node C (see Algorithm 1). Similarly, F either sends the search result packet(s) to follow only F-D-B-A or to follow both F-D-B-A and F-D-C-A. Please note that the search result packet sent to follow path F-D-C-A does not arrive to node A, due to filtering (see Section III-A5). Also note that the PC values for each path is as follows; C-A: 49, C-D-B-A: 55, F-D-B-A: 19 and F-D-C-A: 49. Hence the file is downloaded from node F on path F-D-B-A, if single source is preferred, or it is downloaded from nodes C and F on paths F-D-B-A and C-A, if parallel download is preferred. The constants are chosen as $C_h = 3$, $C_b = 0.4$, and $C_t = 0.06$ in this scenario as they are determined through the extensive real life experiments.

C. Download

Our download scheme benefits from the routes set up during the forward and backward path establishment phases of the search operation. Using the availability parameter, the routes are set up according to the quality of the path. Therefore, after the search operation, each node on the download path has the best possible next node information in its routing table for the corresponding download operation.

The node initiating the search operation does not start the download operation as soon as it receives the first search result packet, rather it waits for a period of time. During this period, we assume that the search result packets from relatively higher quality paths may also arrive. Therefore, after this waiting period, the download begins using the highest quality paths established during the search operation.

It is important to note that the download of a file takes place by the use of TCP. Since during our search protocol the necessary routing table entries are set, the reliable transmission of the files can be performed by TCP.

1) Download by Partitioning Scheme: The download operation supported by Wi-Share uses the partitioning concept. Whenever a file is to be downloaded, it is divided into a number of partitions which is determined by its size. The reason for partitioning is to reduce the amount of loss when a download from a node is interrupted. If the files were not partitioned, in case of a disconnection, the whole file would require to be retransmitted. But with the use of partitioning, in such a case, only the partitions that have not transferred yet require to be transmitted. Partitioning also enables download of a file from different sources on different paths in parallel.

2) Periodic Search Requests During Download: During a download operation, the downloading node issues periodic search requests for the file being downloaded. Benefit of these requests is twofold: First, the availability information in the search and result packets enable update of the routing tables using the most recent availability parameters of the nodes. Hence, the best routes are chosen during the download. Second, if a new node sharing the file being downloaded has joined to the network after the download started, it sends search result packets in order to contribute to the ongoing file transfer. Participation of a new node to an ongoing download is possible due to the partitioning scheme as well as periodic search requests.

3) Route Maintenance: Wi-Share requires that each routing table entry has a timeout value. The timeout value is refreshed periodically if there is an active download using that routing table entry. If no active download is performed using a given routing table entry, then that entry is deleted when the timeout value reaches zero.

Wi-Share relies on the TCP communication protocol during the download operation as explained in Section III-C. Therefore when a connection breaks during the download operation, the two end point nodes of the download operation are informed about the disconnectivity by the use of TCP timeout mechanism. The routing table entries of intermediate nodes that were used by the disconnected download are deleted automatically after the timeout.

When the download operation of a particular file cannot be performed from any node in the ad hoc network, Wi-Share requires that the search operation is initiated. This will ensure that new paths to destination will be found that will better reflect the current status of the ad hoc network. However, during a download operation, which has multiple sources, in case of disconnections from the sources if at least one connection is alive, Wi-Share benefits from the periodic search request mechanism as explained in Section III-C2.

IV. Experiment Results

In order to evaluate the performance of the proposed protocol, several experiments were conducted. Since the proposed protocol was implemented along with a front-end application, the experiments were carried out on a testbed, which consists of nine notebook computers. The testbed was constructed in an area covering the 3rd, 4th, and 5th floors of the Bilkent University, Engineering Building. The nodes (i.e. notebook computers) in the network were carefully arranged to have the topology depicted in Figure 6. Increasing the number of nodes in the network resulted in more connections, reducing the average length of paths between the nodes. Therefore, the number of nodes in the network was kept at nine, which was found to be the optimum for the given area. It should be noted that many other 802.11b/g networks were active in the same vicinity, which probably affected the performance of the experiments. But in order to minimize the deviations the experiments are held on a weekend, since the 802.11b/g activity was expected to be less compared to a weekday. Same experiments were repeated several times and the averages of the results of those experiments were assumed to be the final results. In the following experiments the default values for the constants are chosen as follows: $C_h = 3$, $C_b = 0.4$, and $C_t = 0.06$. The constants that are chosen are determined through the extensive tests performed in the real-life environment with the developed Wi-Share application. During the tests it is observed that these constants are efficient in making the best routing decisions. It should again be noted that a smaller PC values corresponds to a higher quality path and a path with the smallest PC is selected as the primary path.

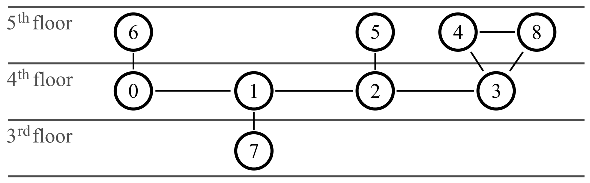


Fig. 6. Topology of the experimental network

Mainly two groups of experiments were carried. The first was related to the search performance and the second was related to the download performance of the protocol, both of which are described in the following subsections. In order to perform controlled search and download, each node in the network contained a file having a unique name with a common prefix (e.g. “Node0”, “Node1”, “Node2”, etc.). Hence, search or download of a file from a specific node was possible using the unique filename and similarly search or download of a file from all the nodes was possible using the common part of the filename.

A. Search Performance

First, the searches were initiated from Node 6 using keywords targeted for specific files. Therefore, each search caused a reply from a single node. The average reply arrival times varied between 0.2 and 0.9 seconds which were almost linearly proportional to the number of hops between the node that initiated the search (i.e. Node 6) and the node that replied. The average reply arrival times are shown in Figure 7. Please note that, in the figure different nodes having the same hop distance from Node 6 are shown separately. In the second experiment, the search was again initiated from Node 6. But different from the first experiment, the keyword was chosen such as (i.e. “Node”) the search caused replies from all the nodes. As shown in Figure 7, this time the reply arrival times increased slightly, where the longest arrival time was around 4 seconds. The main reason for the increase in the arrival times is the increased amount of search replies which increased the bandwidth usage and possibly caused more collisions with other search replies and search requests.

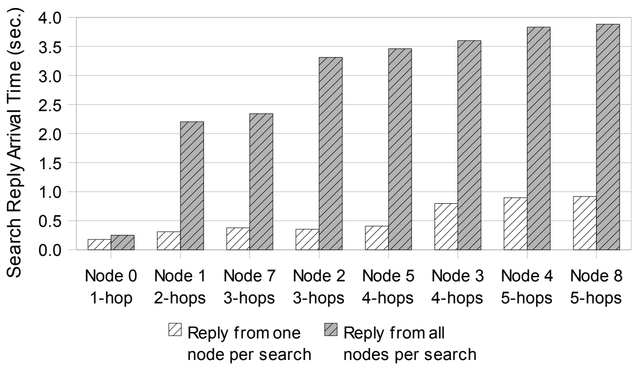


Fig. 7. Search reply arrival times

The overhead due to the searches were also observed in the experiments. Search request packets, which contain the search IDs, keywords, path and availability information, and search reply packets, which contain IDs of the corresponding searches, search results (i.e. file names), path and availability information, constitute the overhead. Figure 8 depicts the search overhead for the first search experiment in which a single node replied for each search request. The search overhead was measured between about 14 and 35 KB and it was observed that in general, as the hop count between the node initiating the search and the node replying the search, the overhead increased.

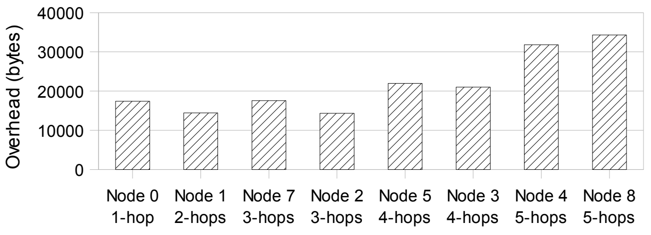


Fig. 8. Search overhead

B. Download Performance

In the download experiment, download of a file, which is 1 604 376 bytes (≊1.53 MB) long, was initiated from Node 6. Similar to the search experiments, each node had the same file with a unique name, hence the source node could be controlled. For example, if the performance of a download from Node 0 was under examination, the file with the name "Node0" was downloaded. Figure 9 depicts the download times of the file from 1-, 2-, 3-, 4- and 5-hops ahead, which correspond to nodes 0, 1, 2, 3, and 4 respectively.

V. Conclusion and Future Work

The routing protocol that is proposed in this paper is aimed to support an ad-hoc network which is used for a P2P file sharing system. During the search phase of the protocol, the necessary entries are inserted to the routing table in the intermediate nodes. The routing decisions are determined upon the availability parameter of the path which identifies the path's usability in terms of the power resources of the nodes, number of hops to be traversed, and the data traffic. By the use of availability parameter and filtering, the best paths are built during the search phase. The download phase of the protocol benefits from the use of download partitioning scheme and contribution of newly joined nodes to ongoing data transmissions. By using such a search and download mechanism our protocol is aimed to work efficiently for ad-hoc networks supporting P2P file sharing systems.

As for the future work we will be performing comprehensive simulations along with the testbed to further observe the performance of Wi-Share.

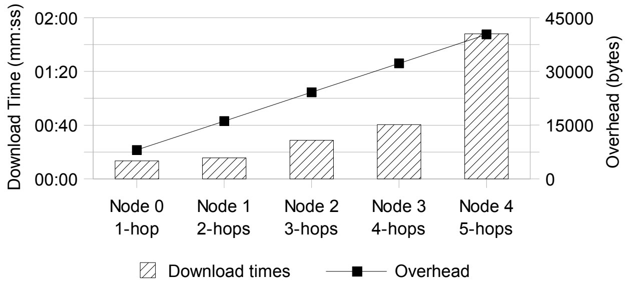


Fig. 9. File is downloaded from one node at a time

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] D. Johnson and D. Maltz, Dynamic source routing in ad-hoc wireless networks, in Proceedings of SIGCOMM96, ACM, California, USA, August 1996.
[4] C. Perkins, E. Belding-Royer, and S. Das, Ad hoc On-demand Distance Vector (AODV) routing, July 2003, RFC 3561.
[5] A. Klemm, C. Lindemann, and O. P. Waldhorst, A special-purpose peer-to-peer file sharing system for mobile ad hoc networks, in Vehicular Technology Conference, 2003, vol. 4, October 2003, pp. 27582763.
[6] R. Draves, J. Padhye, and B. Zill, Routing in Multi-Radio, Multi-Hop Wireless Mesh Networks, ACM MobiCom, Sept. 2004, pp. 11428.
[7] D. Passos et al., Mesh Network Performance Measurements, Intl. Info. and Telecommun. Technologies Symp., Dec. 2006
[8] C. E. Koksal and H. Balakrishnan, Quality-Aware Routing Metrics For Time-Varying Wireless Mesh Networks, IEEE JSAC, vol. 24, no. 11, Nov. 2006, pp. 198494
[9] M. Conti, E. Gregori, G. Turi, A Cross-Layer Optimization of Gnutella for Mobile Ad Hoc Networks, MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing., pp. 343 - 354 2005
[10] H. Sözer, M. Tekkalmaz, and I. Korpeoglu, A Peer-to-Peer File Search and Download Protocol for Wireless Ad-Hoc Networks, Computer Communications (2008), doi: 10.1016/j.comcom.2008.09.004