Improving Gnutella Query Search Algorithm with Jumps

Kholoud Althobaiti[1], Sara Jeza Alotaibi[1], Hend Alqahtani[2]
Taif University[1], Saudi Arabia
Umm Al-Qura University[2], Saudi Arabia

Abstract

The measurement of a search algorithm for unstructured P2P network centres on the number of nodes not receiving their requested files (number of failures) and the number of hops per query. Most current search algorithms are unable to guarantee the success of the query. This study involves a comparison of the strengths and weaknesses of three algorithms of Gnutella P2P protocol, namely Flood, Random Walk, and Random Walk with Neighbours Table. Based on this comparison, a new query search method—referred to as Random Walk with Jumps—is proposed. The experiment proves that the proposed algorithm can obtain a better result with a small number of failures and a minimum number of hops.

1. Introduction

Peer-to-Peer systems (P2Ps) have emerged as a big social and technical event over the past 15 years [1]. Two key reasons have promoted the rapid growth of such systems, namely the low cost and the large number of storage resources on the one hand, and the increased network connectivity on the other hand [9]. Therefore, the P2P network has been gaining in popularity over recent years.

Peer-to-peer networks generally implement some form of virtual overlay network on top of the physical network topology, where the nodes in the overlay form a subset of the nodes in the physical network [4]. Based on how the nodes are linked to each other within the overlay network, and how resources are indexed and located, we can classify networks as unstructured or structured. In an unstructured P2P network, such as Gnutella, no rule exists that defines where data is stored, whilst in a highly structured P2P network, such as Chord, the network architecture and the data placement are precisely specified. The neighbours of a node are well-defined and the data is stored in a well-defined location [2].

Peer-to-peer networks can also be classified into centralised and decentralised. In a centralised network, such as Napster, the system makes use of some form of central server that acts as a broker between peers. The unique server gathers information (only indexes) about the clients. In contrast, decentralised systems, such as Gnutella, use networks of interconnected servers, replacing the unique server of centralised architecture; all users (peers) can be both clients and servers [4].

One of the challenges in P2P networks is searching the content of nodes (files). In pure P2P systems, individual computers communicate directly with one another and share information and resources without using dedicated servers. In essence, this is more like a self-organised network of independent entities.

In this paper, we conducted a detailed study of the search methods of a pure P2P system: Gnutella. Gnutella protocol is a decentralised, unstructured peer-to-peer system, consisting of nodes connected to one another over TCP/IP network topology and running software that implements the Gnutella protocol [2]. Gnutella primarily is used for file-sharing and has been recognised as a popular file-sharing protocol in P2P networks [2].

The file-sharing system in the Gnutella network operates as follows:

1. A user (Node A) starts with a networked computer that runs one of the Gnutella clients. The user then connects to another Gnutella networked computer (Node B). Subsequently, Node A announces its existence to Node B.
2. Node B then announces to all its neighbouring nodes that Node A is existent.
3. Once the rest of the nodes are aware of the existence of Node A, the user at Node A is then positioned to query the data shared across the network. [6]

We benefited from Gnutella's large existing user base and open architecture. We compared three search methods of the P2P system Gnutella, namely Flood, Random Walk and Random Walk with Neighbours Table. The measurements and the analysis of the three methods of Gnutella are carried out when the number of nodes is variable. They are driven by two primary questions:

1. Which of the three methods has the least average hops and/or number of failures, and which has the highest?
2. What is the impact of expanding the network on the success of the query?

In light of the outcomes, we improved a new search method - referred to as Random Walk with Jumps - that allows for better scaling and increased reliability.

Copyright © 2015, Infonomics Society

The rest of the paper is structured as follows: Section 2 presents the background and the Literature Review, whilst Section 3 describes the comparison of the three methods, which helps to answer the questions introduced earlier; Section 4 introduces the new query search method for Gnutella; the results and testing are given in Section 5, whilst Section 6 presents the conclusion and suggestions for future work.

2. Background and Literature Review

The aim of this paper is centred on identifying the challenges that still exist in Gnutella P2P query methods. There are many studies concerned with improving the Gnutella network: in 2003, researchers Tsungnan and Hsinping concluded that the Flooding algorithm generates the best performance in terms of Search Responsiveness, but its Query Efficiency is low due to a huge number of redundant messages. The Random Walk algorithm enjoys high query efficiency, but nonetheless suffers from low search responsiveness [14]. From other points, the study concludes that one of the advantages of Random Walk over Flooding in unstructured overlays is that the former provides more precise control over the number of overlay nodes visited to satisfy a query. [15]

However, this study differs significantly from past works, both in purpose and detail. The researchers used an existing Gnutella code to analyse the current query methods in order to assess their overall ability to satisfy the file search requirements. Then, based on the results, the researchers developed an improved query method with a small number of failures and a minimum number of hops.

The study started by examining the Gnutella code [3], with the reason behind using this source owing to the fact that it is open source and written in Java programming language, which is the preference amongst researchers.

3. Comparison of Gnutella Methods

A good search method must have a small number of failures and a minimum number of hops. This section provides a comparison of three Gnutella methods, namely Flood, Random Walk and Random Walk with Neighbours Table, using an existing Gnutella open source code to determine the value of each search method.

The comparison has garnered the results of counting the average hops and the number of failures for each of the three methods. The goal of analysing these methods is centred on answering the questions posed earlier in Section 1: Which has the least average hops and/or number of failures, and which has the highest? Moreover, what is the impact of expanding the network on the success of the query? Answering these questions will prove pivotal in evaluating the search quality of each of the three Gnutella algorithms.

The study focuses on calculating the number of failures and the average hops resulting from 10,000 runs in an overlay of 10,000–100,000 nodes, taking into account the average of all runs. Some of the values have been fixed in an effort to establish accurate results. The maximum number of neighbours is 10; the number of files is 9,000 (with a maximum number for each node between 5 and 50). The rest of this section highlights the advantages and disadvantages associated with each of the three methods discussed as follows.

3.1. Flooding Algorithm

Flooding distributes the file across every graph in a network. The following steps depict how this algorithm works:

1. One of the nodes requests a file.
2. Each node works as a sender and receiver, except in the case of the first node.
3. Each node attempts to send all messages to all its neighbours, with the exception of the source of the message. Therefore, at the very end, the querying node will receive its requested file, as seen in Figure 1 [4].

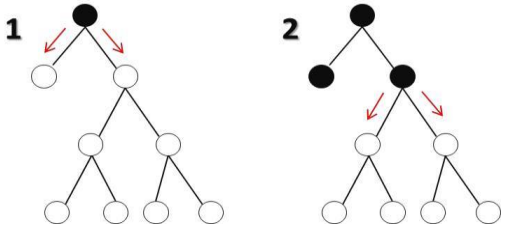

Figure 1. Flooding algorithm

The benefits of this algorithm are: if the packet can be delivered, it will be. Since flooding uses all existing paths, it also uses the shortest path. Flooding is easy to implement [4]. However, Flooding Algorithm is costly since it wastes bandwidth. The message supposed to arrive at one specific node but instead it sent to all nodes [4]. In addition, duplicated deliveries may occur, which could drive the algorithm to infinite loop unless certain precautions are taken. For example:

1. Allow each node to keep track of every packet seen, and then forward each of those packets only once.
2. Enforce a network topology without loops (P2P networks like Gnutella do not have topology, meaning there is no need to take this precaution).

Copyright © 2015, Infonomics Society

```
//Flood asks every neighbor if they have the
file
public int Flood(Query in){
   boolean fileFound = false;
   //at the beginning it is not found
   in.nodeIdsVisited.add(new NodeIds(nodeId));
   //visit other nodes
   in.hopCount++;

if(this.fileList != null){
//if the node's file list not empty
   for(int i=0; i< fileList.size(); i++){

if(fileList.get(i).FileName.equals(in.file.FileN
ame) ) //search for the file
   fileFound=true;} //file found
        }
   }

if(fileFound){
   return nodeId; //found with nodeId
   }
else if(in.hopCount >= 7){
   return -1; //not found -1 hopecount
   }
else{
   if(neighbors != null){
   //visit another neighbor
        for(int i= 0; i< neighbors.size(); i++){
//zero node is special node and does not exist
if(neighbors.get(i).nodeId !=0){
   return neighbors.get(i).Flood(in);}
        }
   }
}
   return -100 //fail to found the file
}
```

3.2. Random Walk Algorithm

The Random Walk Algorithm is a popular alternative to Flooding. It distributes the file to a Random walker, which chooses a random node to walk to. It is considered the best choice in applications for statistics, Physics and Artificial Intelligent (Bayesian Inference). It is also referred to as Markov Chain [16].

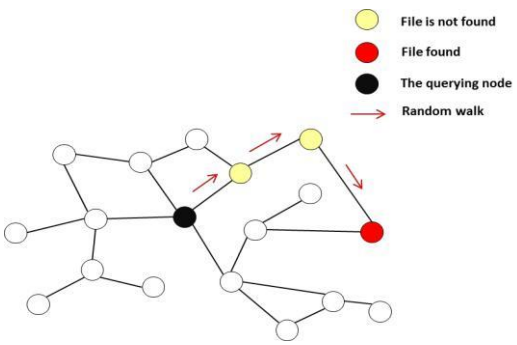

Figure 2. Illustrate Random Walk algorithm

This Algorithm is explained in Figure 2, with the help of the following steps:

1. The querying node sends the number of queries to a randomly selected neighbour. Each one of these queries is referred to as Random walker.
2. Each Random walker has a hop count (in this study, its value is 7). When a node receives a random walker, it searches for the requested file in the node file list. If the file is not found, the node checks the hop count. If the hop count=>0, it decrements by one and forwards the query to a randomly chosen neighbour. If count=0, the query is not forwarded. On the other hand, if the file is found, the query is not forwarded and a reply is sent to the querying node.

```
//random walk uses a random walker who chooses a
random node to walk to
Public int RandomWalk(Query in){
//pick random neighbor to ask
Random rand = new Random();
int neighborToAsk = 0;
in.nodeIdsVisited.add(new NodeIds(nodeId));
in.hopCount++;
boolean fileFound=false;

if(neighbors == null || (neighbors.size() <=0)){
   return -100; //No neighbors mean fail
   }
else{
//remember the neighbor is a random
neighborToAsk = rand.nextInt(neighbors.size());
   }
if(this.fileList != null){
//search for file
   for(int i=0; i< fileList.size(); i++){
        if(fileList.get(i).FileName.equals(in.fil
e.FileName) ){
          fileFound=true;
          }
       }
   }
}

if(fileFound){
   return nodeId; //it found at nodeId
   }
else if(in.hopCount >= 100){
   return -1;
   }
else{
   if(neighbors != null){
return
neighbors.get(neighborToAsk).RandomWalk(in);
       }
}
return -100; //fail to find the file
}//end method Random Walk
```

Copyright © 2015, Infonomics Society

Random Walk algorithm is equally successful to ordinary Flooding. If the file is nearby, it can be considered less costly than Flooding [17–18]. It does not require any topology information; P2P network does not have any topology network [18]. The performance of the Random walk depends on choosing the parameters of the Random walk (the number of queries and hop counts). The popularity of a resource is required for the parameters selection module to set their values. The number of requests here is fewer than in an ordinary Random Walk [17].

After all, Random Walk Algorithm has several problems. If the file is not found, more messages are generated than in the case of Flooding [18]. Each random walker needs a time allowance to live or hop count; otherwise, duplicated deliveries could occur. Choosing low values for parameters for searching a file with a low popularity estimate would result in a low success rate and high delays, whilst choosing high values of parameters for searching a file with a high popularity estimate would result in excessive overhead [17–18].

3.3. Random Walk with Neighbours Table Algorithm

This is similar to ordinary Random walk; in this method, the querying node checks to determine whether its neighbours' list of files has the desired query within them. If so, that neighbour is walked to if it is not randomly chosen.

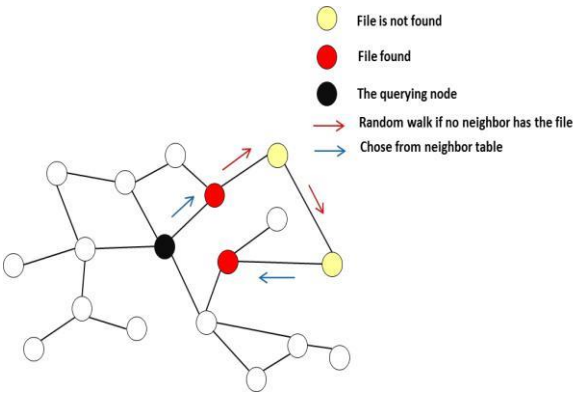

Figure 3. Illustrate Random Walk with Neighbours Table algorithm

As shown in Figure 3, the algorithm works as follows:

1. The method will begin by examining the querying node's list of neighbours. It searches for the requested file in each neighbour's list of files. If the file is found in the file list of any neighbour, the method walks directly to the neighbour. If the file is not found on any list, a neighbour is randomly chosen, and the method completes another search for the file in the neighbour's list.
2. Each one of the randomly generated has a hop count (in this code, its value is 7). When this node receives the request, it searches for the requested file in this node's list of files. If the file is not found, the node checks the hop count. If hop count=>0, it decrements by one and forwards the query to another randomly chosen neighbour. If count=0, the query is not forwarded. On the other hand, if the file is found, the method walks directly to that neighbour.

There are many advantages of this algorithm. If the requested file is nearby, the method is considered less costly, even when compared with Ordinary Random Walk. It does not require any topology information; P2P networks do not require any [19–20].

However, there are limitations to the algorithm: for example, if the file is not found, more messages are generated. Each random move requires TTL or hop count; otherwise, duplicated deliveries could occur [19–20].

```
public int RandomWalkWithNeighborTable(Query
in){
//pick random neighbor to ask
Random rand = new Random();
int neighborToAsk = 0;
boolean neighborHasFile=false;
int neighborWithFile =0;
in.nodeIdsVisited.add(newNodeIds(nodeId));
in.hopCount++;
boolean fileFound=false;
//Begin Lookup of neighborTable
if(neighbors != null){
for(int i= 0; i< neighbors.size();i++){
   if(this.fileList != null){ //make sure
for(int j=0; j< fileList.size(); j++){
//look for the file in fileList
if(fileList.get(j).FileName.equals(in.file.FileN
ame) ){
       neighborHasFile=true; //file was found
       neighborWithFile = i; // assign the value
to visit
break;
}} } }}
if(neighbors==null||(neighbors.size()<=0)){
   return -100; //No more Neighbors.fail
   }
else if(neighborHasFile){
   neighborToAsk = neighborWithFile;
   }
else{
   neighborToAsk=
   rand.nextInt(neighbors.size());//ask again
    }
if(this.fileList != null){
// if it hold files then will search in its list
for(int i=0; i< fileList.size(); i++){
if(fileList.get(i).FileName.equals(in.file.FileN
ame) ){
   fileFound=true;
       }
   }
```

3.4. Results of Analysing Gnutella Methods

In general, the results of the analysis show that number of failures is the least in Flooding compared to Random Walk and Random walk with Neighbours Table. Although Flooding has the lowest number of failures (strength), as shown in Table 1 and described in Figure 4, at the same time, it also has the highest average of hops (weaknesses). Random Walk shows the lowest average hops (strength) and a moderate value for the number of failures, as mentioned in Table 2 and Figure 5. The results of the Random Walk with Neighbours Table do not differ significantly from those of the Random Walk method.

Table 1. Analysing result of no. of failures

	Flood	RW	RW with NT
10000	1153	3733	4489
20000	933	4096	4123
30000	638	3526	3949
40000	583	3581	4018
50000	528	3598	3611
60000	561	2750	3404
70000	419	3560	3669
80000	250	3214	3602
90000	295	3400	3231
100000	387	3221	2817

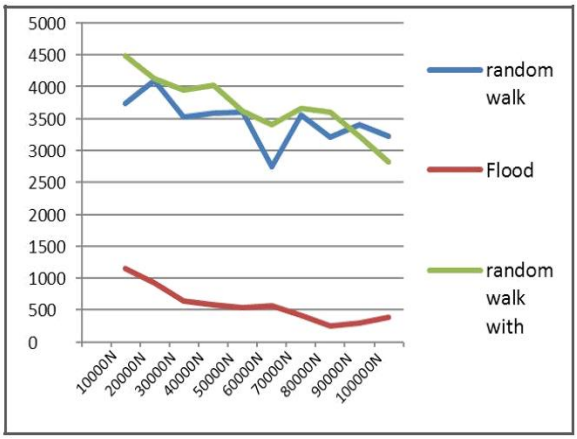

Figure 4. Analysing result of no. of failures

Table 2. Analysing result for average hop

	Flood	RW	RW with NT
10000	4.8500	3.9782	3.9151
20000	4.8830	4.1479	4.0993
30000	4.9899	4.4332	4.2190
40000	4.9982	4.4126	4.2155
50000	5.0326	4.4500	4.4166
60000	5.0109	4.9728	4.6250
70000	5.0067	4.5010	4.4335
80000	5.0455	4.7446	4.4353
90000	5.0302	4.6148	4.7522
100000	5.0468	4.7681	5.0731

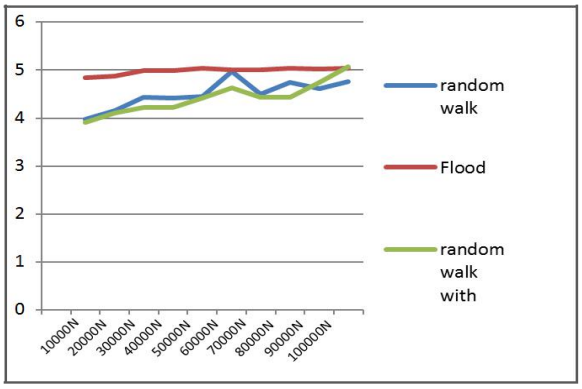

Figure 5. Analysing result of average hop

In light of these results, it can be seen that writing a new method should be done in order to minimise the number of failures as in Flood, whilst at the same time minimising the number of requests as in the Random Walk method.

4. Design New Search Query Method

This paper proposes a new search query method. The so-called Random Walk with Jumps is a popular alternative to Flooding. This algorithm can be considered a Random Walk search algorithm in which the random walk makes jumps. The expected jump length is randomly walked (see Figure 6).

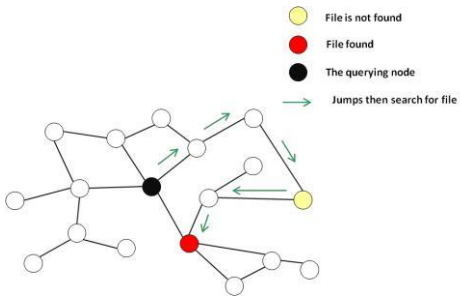

Figure 6. Illustrate Random Walk with Jumps algorithm

The querying node sends the number of queries to a randomly selected neighbour. It then chooses a random length to jump. If the jump length=>0, it passes to another random neighbour and decrements the length by one. If jump=0, it stops passing and searches for the file in the current node. This step then is considered a Single Random Walk.

Each Random walker has a hop count (in this code, its value is 7). When a node receives a Random walker, it searches for the requested file in the node file list. If the file is not found, the node checks the hop count. If hop count=>0, it decrements by one and forwards the query to a randomly chosen neighbour, and accordingly performs another random walk. If count=0, the query is not forwarded.

Copyright © 2015, Infonomics Society

On the other hand, if the file is found, the query is not forwarded and a reply is sent to the querying node.

4.1. Advantages

- As successful as an ordinary Random walk.
- The jumps reduce focus on a specific area. This advantage provides the ability to cover different areas of node locations and, subsequently, speeds up the search and accordingly reduces network capacity [5].
- If the file is on a remote area, the new method can be considered less costly than Random walk [5].
- The method reduces the number of requesting messages, even if there are no files in the graph. The number of requests is smaller than in the ordinary Random Walk [5].

4.2. Problems

As with the Random Walk, if low values were chosen for parameters, searching for a file with low popularity estimate would result in a low success rate and high delays, whilst choosing high values of parameters to search with high popularity estimates would result in excessive overheads [5].

4.3. Code Description

The code below starts with a conditional statement that allows the querying node to check whether there are neighbours in the graph. If the nodes do not have any neighbours, this means the jump is a fail; the value of fail then would change to true.

```
public int RandomWalkWithJumps(Query in , int
length){
Random rand = new Random();//Randoma walk
int neighborToAsk = 0;
in.nodeIdsVisited.add(new NodeIds(nodeId));
boolean fileFound = false;//at the begin no file
found
if(neighbors == null || (neighbors.size()<=0)){
   fail = true; //fail because no neighbors
   }
else{
   neighborToAsk =
   rand.nextInt(neighbors.size());
    }
```

Thus, if the jumps are not yet finished and do not fail, the method would then jump to the next neighbour. It keep jump until jump count finish or no more neighbours.

```
if(length>0 && !fail){
   return
neighbors.get(neighborToAsk).RandomWalkWithJumps
(in,length-1);
   }
```

After finishing the jump, the method searches for a specific file in the file list at the last neighbour in the jump.

```
if(this.fileList != null){
   for(int i=0; i< fileList.size(); i++){
       if(fileList.get(i).FileName.equals(in.fil
e.FileName) ){
          fileFound=true;
          }
       }
   }

if(fileFound){
   return nodeId;
   }
```

The search will check different cases: it will check whether the file is found, and then will return the current node ID if it is. It will also check whether the hop count is already 3; if it is, it will return –1 (small because the graph does not have huge dimensions). If there are neighbours and the jump does not fail, the search will again return to the function. If the case is not one of these, it returns –100, which means fail, and the number of failures increases by 1.

```
if(fileFound){
   return nodeId;
   }
else if(in.hopCount >= 3){
   return -1;
   }
else{
   if(neighbors != null && !fail ){
   length = rand.nextInt(3);
   in.hopCount++;
return
neighbors.get(neighborToAsk).RandomWalkWithJumps
(in , length);
   }
}
return -100;
}//end method Random Walk With Jumps
```

5. Testing and Results

In general, the findings indicate that the value of average hops is approximately 2. In addition, the number of failures is between (1500< #of failures<4500) for 10,000 test times. The number of failures decreased when the number of nodes increased in all four algorithms because the number of files is fixed whilst the number of nodes is variable. Wherefore, the probability of finding the file increased in 100,000 nodes more than 10,000 of them.

As it can be seen in Table 3 and Figure 7, the Random Walk with Jumps algorithm has the best score in average hops as it jumps first then asks for a specific file, meaning it searches a wide area. The number of failures is not as minimised as in Flooding because Flood sends the query for all the nodes. However, Random Walk with Jumps is better than

Copyright © 2015, Infonomics Society

the Random Walk and Random Walk with Neighbours Table (see Table 3, Figure 7). Thus, this method beats the best values in the three methods, which are the number of failures as Flooding and average hop as Random Walk and Random Walk with Neighbours Table. The following results are for the number of nodes between 10,000 and 100,000.

Table 3. Analysing results of four methods for number of failures

	Flood	RW	RW with NT	RW with Jumps
10000	1153	3733	4489	2220
20000	933	4096	4123	2285
30000	638	3526	3949	1939
40000	583	3581	4018	1248
50000	528	3598	3611	1077
60000	561	2750	3404	1110
70000	419	3560	3669	1127
80000	250	3214	3602	647
90000	295	3400	3231	733
100000	387	3221	2817	857

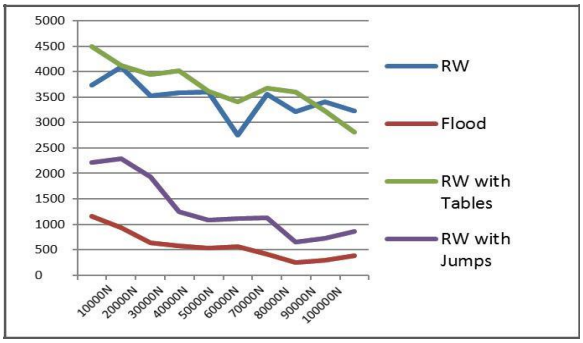

Figure 7. Analysing results of four methods for number of failures

Table 4. Analysing results of four methods for average hops

	Flood	RW	RW with NT	RW with Jumps
10000	4.8500	3.9782	3.9151	2.9979
20000	4.8830	4.1479	4.0993	2.991
30000	4.9899	4.4332	4.2190	2.9906
40000	4.9982	4.4126	4.2155	2.988
50000	5.0326	4.4500	4.4166	2.995
60000	5.0109	4.9728	4.6250	2.993
70000	5.0067	4.5010	4.4335	2.993
80000	5.0455	4.7446	4.4353	2.9843
90000	5.0302	4.6148	4.7522	2.9919
100000	5.0468	4.7681	5.0731	2.9917

Social life, which has promoted the success of the Gnutella network, might change, causing the network to fade. However, P2P is recognised as one of those rare things that, quite simply, are too good to go away.

The open architecture, achieved scale and self-organising structure of the Gnutella network make it an interesting P2P architecture for examination [9]. The measurement and analysis techniques used here also can be used for most P2P systems in order to enhance general understanding of the design.

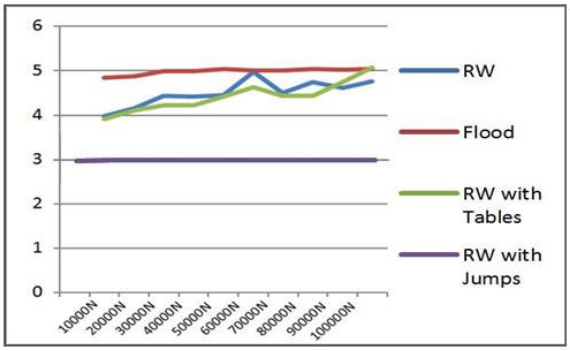

Figure 8. Analysing results of four methods for average hops

This paper analysed the Gnutella network in an effort to study its current methods. The study then compared three of the methods, namely Random Walk, Flooding and Random Walk with Neighbours Table, in an effort to identify which has the least average hops and/or number of failures and which has the highest. Subsequently, the paper proposed a new query search method, known as Random Walk with Jumps. The new method has proven to be a much better approach with a small number of failures and a minimum number of hops.

There is a potential direction for future study, which is centred on improving the new method to achieve a fewer number of failures.

7. References

[1] A. Basu, S. Fleming, J. Stanier, S. Naicken, I. Wakeman, and V. Gurbani, (2013). 'The state of peer-to-peer network simulators‖' ACM Computing Surveys (CSUR), 45(4).

[2] Y. Wang, X. Yun and Y. Li, (2007). 'Analyzing the Characteristics of Gnutella Overlays', Information Technology, IEEE, pp. 1095–1100.

[3] Tsungnan and Hsinping, (2010, May.). 'Gnutella-peersim Experiment with gnutella', Google code, https://code.google.com/p/gnutella- peersim/ (12 May, 2015)

[4] S. Tanenbaum and J. W. David, (2010). Computer Networks, Pearson Publisher, 5th Edition.

[5] R. Beraldi (2009) 'Random walk with long jumps for wireless ad hoc networks', Department of Computer and Systems Engineering, University of Rome, Italy, vol. 32, pp. 294–306.

[6] F. Hermann, (2002). Scalability of the Gnutella Network and Business Opportunities of Peer-to-Peer Networking, GRIN Verlag Publisher.

[7] J. Harris, (2005). 'A Scalable & Extensible Peer-to-Peer Network Simulator', Ottawa-Carleton Institute for Computer Science.

Copyright © 2015, Infonomics Society

[8] M. Baker and R. Lakhoo, (2007). 'Peer-to-peer simulators', ACET University of Reading.

[9] M. Ripeanu, (2001). 'Peer-to-Peer Architecture Case Study: Gnutella Network', The University of Chicago.

[10] P. Murthy, (2003). 'GTKgREP - Design and Implementation of a Gnutella-based Reputation Management System', North Carolina State University.

[11] V. Aggarwal, A. Feldmann and S. Mohr, (2005). 'Implementation of a P2P system within a network simulation framework', Technische Universität München, Germany.

[12] S. K. Dhurandher, S. Misra, M. S. Obaidat, I. Singh, R. Agarwal & B. Bhambhani, (2009). 'Simulating Peer-to-Peer networks', pp. 336–341, IEEE/ACS.

[13] G. Yutang, L. Wanli & L. Bin, (2007). "Improved Resource Discovery Algorithm on Gnutella Based on P2P Networks', Control Conference, p. 599, IEEE.

[14] T. Lin and H. Wang, (2003). 'Search Performance Analysis in Peer-to-Peer Networks', National Taiwan University.

[15] M. Castro, M. Costa and A. Rowstron, (2003). 'Should we build Gnutella on a structured overlay?', Microsoft Research, Cambridge.

[16] A. Prakash, (2006). 'A Survey of Advanced Search in P2P Networks', Department of Computer Science, Kent State University.

[17] Keqin Li, (2010). 'Performance analysis and evaluation of random walk algorithms on wireless networks', Parallel & Distributed Processing, Workshops and Phd Forum, IEEE, pp. 1–8.

[18] J. Cui, J. Guo, C. Zhang & X. Chang, (2012). 'Implementation of random walk algorithm by parallel computing', 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), IEEE, pp. 2477–2481.

[19] D. Stutzbach, R. Rejaie. N. Duffield, S. Sen and W. Willinger, (2009). 'On unbiased sampling for unstructured peer-to-peer networks', IEEE/ACM Transactions on Networking (TON) Journa, 17(2), IEEE/ACM, pp. 377–390 .

[20] Gkantsidis, Christos, Milena Mihail and Amin Saberi, (2004). 'Random walks in peer-to-peer networks' Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies. IEEE.

Copyright © 2015, Infonomics Society