Journal of High Speed Networks 23 (2017) 287–309
DOI 10.3233/JHS-170572
IOS Press

A scalable and hierarchical P2P architecture based on Pancake graph for group communication

Abdelhalim Hacini [a], Mourad Amad [a,b,*] and Semchedine Fouzi [a,c]
[a] *LaMOS Research Unit, Computer Science Department, Faculty of Exact Sciences, Bejaia University, 06000 Bejaia, Algeria*
[b] *Bouira University, 10000 Bouira, Algeria*
[c] *Setif University, 19000 Setif, Algeria*

Abstract. P2P network is characterized by several important aspects: scalability, decentralization and dynamicity. One of the major problems posed by decentralization and dynamicity of P2P network is how to discover and access to a resource, since it is very difficult to get an overall view of all resources in the P2P community. The most important concern is how to locate a particular resource and identifying the optimized path, that reaches the peer responsible for this requested resource, with a minimum number of hops and an optimized delay.

In this paper, we propose a new scalable and hierarchical P2P architecture for lookup acceleration and optimization based on Pancake graph. This architecture can supports many type of applications such as: file sharing, collaborative work and asynchronous distributed group communication, etc. The performance evaluation shows that results are globally satisfactory.

Keywords: P2P network, Pancake graph, lookup optimization, group communication

1. Introduction

Among the fundamental problems of peer to peer networks is to find the peer that stores the profiles of the users (*resources*). This problem is complicated; because in peer to peer networks there is no central server and the volatility of the peers is rather high (*nodes can leave the network constantly as others can join it*). The most executing function in P2P network is without doubt the lookup process. The main objective is then how to optimize the number of hops carried out by the lookup algorithm (*routing at application layer*), to reach the resource in a less time, and not vainly overloading the network by the overhead messages. On the one side, it is necessary for each source node that looks for a resource to identify the assigned corresponding key and on the other side, to find the peer responsible for this key (*the destination node*) with the optimized number of hops.

In this paper, we exploit and we analyze the properties of Pancake graphs, in order to give a new lookup model for P2P networking, based on the notion of permutation to identify the nodes. The rest of this paper is organized as follows: in Section 2, we present a background on P2P network, follow-up by some works studying the topic of lookup problem in P2P networks, especially which are built over particular graphs in theirs architectures and processes of routing. Also, we talk about Pancake graph based solutions, which can be considered as an excellent graph conceived as a solution for the lookup problem. Section 3 presents details of our proposition such as: the underlying architecture, the lookup process, the description of routing table and the node joining and leaving processes. Section 4 gives the performance evaluation of the proposed solution in terms of number of hops and also the delay. We compare our results to those of existing solution based on the Pancake graphs. Finally, we conclude and give some future works.

*Corresponding author. Tel.: 00213551428098; E-mail: amad.mourad@gmail.com.

2. Background and related work

In this section, we discuss the area of P2P networks through: its definitions, proprieties, applications, architectures and some of particular structures of graphs such as: De Bruijn graphs, Skip graphs, Knodel graphs, especially the routing process applied in each kind of graph. Also, we discuss the utilization of Pancake graph as underlying P2P architecture, which is founded on the notion of permutation for node identifications.

2.1. Background

P2P concept represents a paradigm shift from the client-server model to a more decentralized model, in order to extend the limits imposed by traditional distributed systems. So, it goes much further the file-sharing applications. While allows to decentralize services and to place resources at the disposal of any user in the network. Among the main properties of this type of network there are: no centralized coordination, no centralized database, no node has a complete view of the system. The total behavior starting from the local interactions. All the services and data are accessible from any node. The nodes are autonomous.

P2P systems are used in several categories of applications, which can be divided into fourth main classes: file sharing, distributed computing, communication and collaboration [3]. According to the various levels of decentralization, P2P architectures are also classified into three principal categories: centralized, completely decentralized (*unstructured, structured*) and hybrid architecture [4]. Recent research in P2P systems has been focused on three categories of P2P systems, offering [17]:

- Communication and collaboration,
- Distributed computation,
- Content distribution.

In the axis of communication, we address the lookup process optimization, particularly how to identify the requested resource with a minimum number of hops, so, structured topologies have been considered and well-studied. Recently, specific topologies such as: De Bruijn graphs [7], Skip graphs [5], Knodel graphs [12] and Pancake graphs [32] are investigated. Our proposition completes these works. The advantages of these graphs is that they offer good properties, such as limited and controlled number of hops required to route a request from source node to destination node and the degree of connectivity of the topological graph. The graph of Pancake takes a very important place in several scientific disciplines, we can quote as examples: Mathematics, Management of the data-processing networks and Molecular biology [22]. Also, it has very important aspects as scalability propriety (*expansion, reduction*), aggregation of data, token distribution on the grid level and appropriate of disjoint paths, which is used in the state of fault tolerant nodes. For this, we use this kind of graphs in order to build a new optimized P2P architecture.

Pancake graph has been proposed by Akers and Krishnamurthy as an alternative of the Hypercube graphs for the interconnection of the processors in parallel computers [25]. It is a graph like the others, holds vertex and arcs, it gathers groups of nodes which are similar to crepes (*Pancakes*) from where its name is derived. It is a directed graph, symmetrical and recursive. It is based essentially on the principle of permutations of the symbols component of its peers [1]. Figure 1 illustrates some Pancake graphs.

The resources in a P2P system are distributed among all peers. These resources must first be found (*and therefore sought*) to access it. A key identifies a resource in a P2P system. The most general form that can be found in the literature is character string or a numerical value of fixed size. Depending on how a P2P system looks resources using these keys and depending on the relationship between the keys and peers, the type of P2P system is defined differently.

The data routed through P2P networks are queries, each peer knows a small number of peers (*compared to the total number of nodes in the P2P network*) called neighboring peers. Peers and connections to its neighbors form the graph $G(x, u)$ which is the P2P network of the substrate. However neighbors of a given peer (*and so the links u*) are chosen according to the lookup algorithm used in the P2P network. So, this lookup algorithm defines the

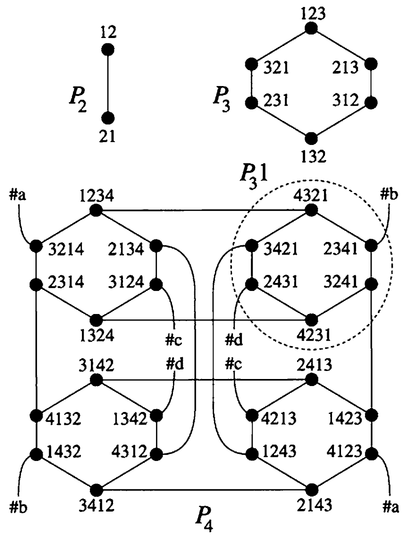

Fig. 1. Pancake graphs P_2, P_3 and P_4.

P2P network, and then determines the graph $G(x, u)$ that modeled it. The set of neighbors of a given peer is called routing table, for what the next hop of a request is decided among these neighbors peer.

Peer to peer systems have more difficulties than client server systems to disseminate information and coordinate interconnection between nodes, thus, ensuring low-latency requests. Therefore, P2P networks that impose structure between connection of nodes, to ensure low communication delays, it is the structured P2P systems. These systems are inspired from the structures of graphs to interconnect the nodes.

2.2. *Related work*

Among the process of routing in P2P networks based on particular graphs, which locate resources with minimum number of hops and delay we can found: graphs of De Bruijn, Skip graphs, Knodel graphs and graphs of Pancake. In the following, we give some works based on these types of graphs.

In the recent years, several P2P systems based on the De Bruijn graphs have been explored. As an example, we can quote:

- The Koorde protocol [20] which extends that of Chord [31] to reach the performances of a graph $B(2, L)$ of De Bruijn. It is one of DHTs which represents the lowest rate of the latency in the absence of congestion,
- The Broose protocol [15], which adapts that of Kademlia [27] to the topology of De Bruijn,
- The D2B protocol [14] which adapts that of CAN [30] to the graphs of De Bruijn,
- Optimal Diameter Routing Infrastructure (*ODRI in summary*) [26] for the networks with constant degree,
- The protocol in [2] optimizes the lookup process in a dynamic environment.

The routing process using De Bruijn graph is as follows: to route a message from a node $X = x_1x_2 \ldots x_m$ to a node $Y = y_1y_2 \ldots y_m$, we make a left shift of the symbols or numbers composing the node source by the symbols or numbers composing the destination node [2]. This technique of routing is optimized in terms of number of hops. The same process previously discussed is executed after each step of hops, follow-up deleting the longest common chain S which is a suffix of x and prefix of y [2]. Graphs with crossing-over or Skip graphs are decentralized and structured P2P architectures, not using a DHTs. They were developed on the basis of list crossing-over or Skip list [28] and there are several variants of them: Skip Net [19], Interval Skip graph [9], Skip Web [24], Rainbow Skip graph [16], Skip Stream [29] and Skip Quadtrees [10]. For this type of graphs, the lookup process begins from the node seeking the key to the highest level. If did not found the key k and that we can go down. The lookup continues at a lower level if appropriate, until it reaches the level 0. As long as the key on the right (*respectively left*) is smaller (*respectively greater*) than k, we move to the right (*respectively left*). Hence an address of the node storing the requested key, if it exists, or the address node storing the key closest to the desired key is returned [5]. Lookup process, insertion and suppression algorithms are described in [5] for more details. Concerning the Knodel graphs which improved theirs efficiency in terms of network Broadcasting and Gossiping, they contain also the propriety of Hamiltonian cycle which is formed by alternating edges in dimension 0 and 1 [18]. For the routing in a P2P network based on Knodel graph $W_{(d,2^d)}$ we distingue [18]:

- The simplest routing is just to travel along the hamiltonian cycle which composed by edges 0 and 1 alternatively,
- A routing in a partial graph which has dimension less than d : dimension 0, 1, and some dimensions between 2 and $d-1$ by checking the following steps:
  - ∗ Determine in which half the destination node is located,
  - ∗ Determine the distance between the destination and the current position along the cycle,
  - ∗ Use the edges of the appropriate dimension and dimension 0 or 1 to get closer the destination,
  - ∗ Use dimension 0 and 1 alternatively to reach the destination along the hamiltonian cycle.
- A routing in a full graph which has d dimensions, by using the binary representation of integer x which composing of d bits (x:*identifier of destination node*), gives a path from source node to the node x, based on the reduction rules introduced in [13].

The routing process based on Pancake graphs has been recently explored. Indeed, according to Y. Suzuki and K. Kaneko [23], an algorithm named *route* has been proposed as a routing process. In *route* algorithm, each peer is represented by a table, it can store a number less than or equal to n (*the order of the graph*). The algorithm gives an elementary path (*which does not uses the same peer more than once*) between two arbitrary peers s and d in a graph n-pancakes [23] (see Algorithm 1).

Algorithm 1 (Routing in a Pancake Graphs)**.**

1. **Procedure Route** ($s = s_1s_2\dots s_n, d = d_1d_2\dots d_n$);
2. **Begin**
3. **For** $i := n$ **to** 1 **step** -1 **Do**
4.   **If** ($s_i \neq d_i$) **Then**
5.   Find k thats $s_k = d_i$;
6.   Select edge $(s, s^{(k)})$;
7.   Select edge $(s^{(k)}, s^{(k,i)})$;
8. $s := s^{(k,i)}$;
9.   **End If**
10. **End For**
11. **End**

Pancake graphs constitute the main axis for several domains of search such as: the Wireless Sensor Networks (*WSN*) [11] and have been discussed for other recently works: [8,21] and [6].

The following table describe and more explain the motivations, the disadvantages of the related works presented and the motivations of our proposed solution (see Table 1).

For that, we offer a new scalable architecture P2P network inspired from the main both axes: the one for the ring architecture which is inspired from the Chord protocol, and the second from the idea of permutation used for the identifier nodes inspired from Pancake graphs with the purpose to optimize both the number of hops and delay for their lookup process for the one to one type of communication, so, we believe that our proposed architecture profit to the advantages of Chord protocol, Pancake graphs and for the lookup process invoked which improved its quality (*optimization for acceleration*).

Table 1
Motivations, disadvantages of related works and our motivations

Type of graphs	Motivations	Disadvantages	Our motivations
De Bruijn graph	– De Bruijn graph is appropriate for maintaining dynamic connections, – Offer an architecture with an interesting connectivity	– The relationship between number of hops of the paths and the diameter of the identifier nodes, – We cannot select other path if one of the nodes which construct it failed or leave the graph.	– Offered an optimized lookup process for P2P networking that leads to the acceleration in terms of both number of hops and delay. – The underlying overlay architecture is improved according to the shortcomings of the other types of graph.
Skip graph	– Skip Graphs supporting complex queries and fault tolerance for even large groups of nodes.	– The duplication of the nodes constructed at all their levels, which implies for the delay of node insertion or node deletion.	
Knodel graph	– Studied as interconnection networks for the good proprieties in terms of broadcasting and gossiping.	– Offer good properties only in terms of broadcasting and gossiping.	
Pancake graph	– For interconnecting processors in parallel computers	– The number of nodes should be equal to $n!$.	

So, our main objective in this paper is to optimize the path relaying any two arbitrary nodes (*source and destination*), that need to communicate between them, using Pancake graph. The optimization is in terms of number of hops and delay, in order to accelerate the lookup process.

3. Proposed solution

In this section, we introduce our proposed solution; we describe the manner of naming nodes and also resource keys, the process of attribution and localization of the keys, the underlying architecture, the maintenance processes, the use of finger table, the lookup process and follow-up by a comparison of the *Route* routing algorithm of Pancake graphs with our proposed algorithm at the implemented P2P network. Hence, to optimize both the number of hops and the delay, we use communications between arbitrary nodes.

So, we propose a new P2P overlay architecture which benefit from the notion of permutation and some of properties such as: expansion and reduction which characterise the Pancake graphs that model P2P network communication, in order to reduce the lookup process that are applied.

3.1. Underlying architecture

The proposed architecture is organized as hierarchical rings, the identifiers of the nodes for this proposed network are all the generated permutations of the set: $\{1, 2, \ldots, n\}$. Let $u = u_1u_2 \ldots u_n$ be a permutation of n symbols $1, 2, \ldots, n$. For each i $(1 \leqslant i \leqslant n)$, an operation $u^{(i)}$ is defined as: $u^{(i)} = u_i u_{(i-1)} \ldots u_1 u_{(i+1)} u_{(i+2)} \ldots u_n$ [12], n is selected according to the full number of nodes that compose the architecture. The nodes are dispersed on the ring in ascending way according to their numerical identifications, i.e. from the minimal value to the maximum value provided, only the nodes which have identifiers finished by the same numerical value should appear in the same ring. Let be $s = s_1s_2 \ldots s_n$ and $d = d_1d_2 \ldots d_n$ two nodes, s and d appear in the same ring if and only if $s_n = d_n$. If $s_n \neq d_n$, each one of them belongs to a different ring. The number of rings depends of the total number of nodes in the network. The ring architecture is chosen because it facilitates the manner of addition and deleting of nodes. Figure 2 illustrates the proposed architecture.

3.2. Attribution and localization of the keys

It is already noted that a service can be simulated by lookup a file such as image or video for example. For that, the lookup of a file in a network should be simplified, optimized and accelerated. Among the known methods,

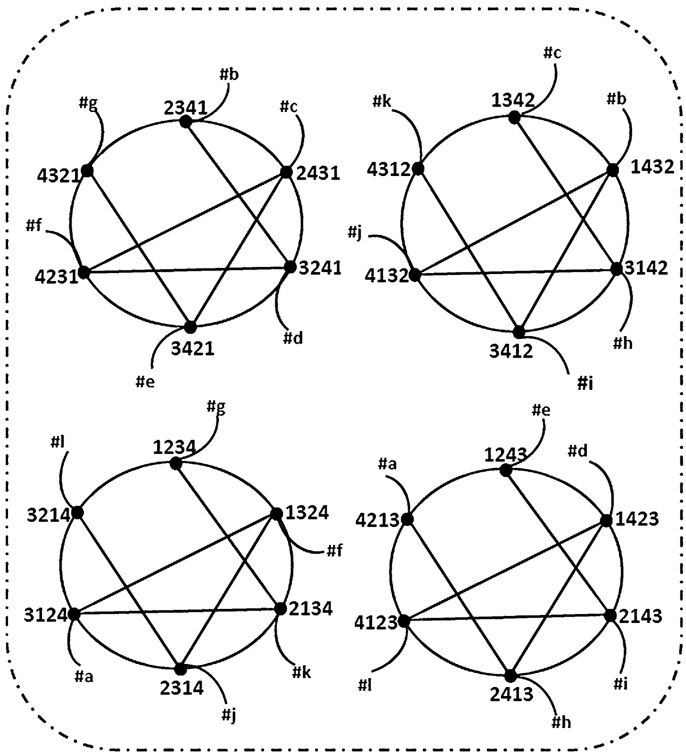

Fig. 2. Architecture overlay of the proposed P2P network.

we find the assignment of keys to files with hash functions in order to better distinguish from each other. As an example, in Chord protocol, the keys are assigned to the nodes according to a specific law (*first higher node for example*). So, to find a given file in such a network, it is enough to use its key to get information about the node identification, which holds it, and to seek this one in order to download it directly.

In the proposed method, we apply the same principle of association of keys used in Chord protocol, such as key values between smallest and highest identifying nodes. So, a key is assigned to the first higher node in term of numerical values provided; that it has a minimum of keys compared to its predecessor (*the predecessor is the node that its numerical identification is less to this higher node*). If this condition is not checked (*highest node has more number of keys to its predecessor node*), the key is allotted to this predecessor node, in order to apply a certain mechanism of load balancing between the nodes of the network, in terms of number of keys which it holds each node. For this reason, we use a variable that has the function of counting the number of keys for each node; in order to allocate the resource to the node that has a minimum number of keys compared to its predecessor.

- **Algorithm of keys attribution**

Algorithm 2 describes the principle of the key attribution used in our proposed solution.

For illustration purpose, we give the signification of variables used in this algorithm in Table 2.

Algorithm 2 (Attribution of the keys to the peers)**.**

1. **Procedure Attribution-Key** ($c_1c_2 \ldots c_n$)
2. *ind*: integer := 1;
3. *Peer*[ ] := the whole of peers $p_1p_2 \ldots p_n$; – tried in an ascending way –
4. *CountKeys*[ ] := count of keys to the each peer of the whole $p_1p_2 \ldots p_n$;
5. **Begin**

Table 2
Variables vs significations

Variable	Signification
$c_1c_2 \ldots c_n$	Identifier of a requested key
Pair[ ]	Table of existing peers
Ind	Index to browse the table
CountKeys	Count the number of existing keys at each peer

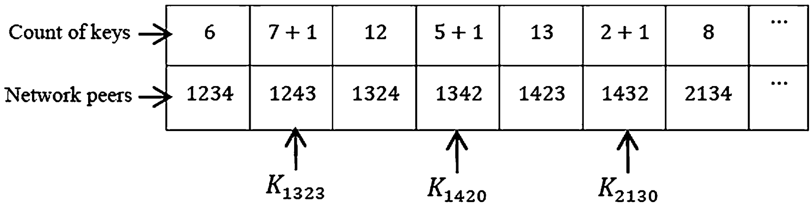

Fig. 3. Examples of attribution of keys.

While ($c_1c_2 \ldots c_n >$ Peer[ind]) **DO**
    $ind++$;
End While
If ($CountKeys[ind] < CountKeys[ind-1]$) **Then**
  $Peer[ind] := c_1c_2 \ldots c_n$;
  $CountKeys[ind]++$;
Else
  $Peer[ind-1] := c_1c_2 \ldots c_n$;
  $CountKeys[ind-1]++$;
Fin If
End

For better illustration, we give the following example:

Example. In Pancake graph p_4, the smallest identifier in term of numerical value is 1234 and the highest is 4321. Hence, the keys are between these two values (*example of keys*: k_{1323}, k_{1420} and K_{2130}...etc) and the whole of peers 1234, 1243, 1324, 1342, 1423, 1432 and 2134...etc, having a number of keys 6, 7, 12, 5, 13, 2 and 8...etc, respectively at a given time. By considering the key k_{1420} as an example, the attribution algorithm finds that the first peer strictly greater than peer 1420 is the peer 1423. It is then noted that it has a higher number of keys (*equal to 13 keys*) compared to its neighbor peer 1342 (*who has only 5 keys*), so the key k_{1420} is allotted to the peer 1342 and raise their key counter from 5 to 6.

- **Algorithm of keys localization**

Algorithm 3 describes the principle of the key localization process used in our proposed solution. We use the same signification of variables as in Algorithm 2.

Algorithm 3 (Localization of the keys)**.**

Procedure Location-Key ($c_1c_2 \ldots c_n$)
ind: integer := 1;
Peer[ ] := the whole of peers $p_1p_2 \ldots p_n$; – tried in an ascending way –
CountKeys[ ] := count of keys to the each peer of the whole $p_1p_2 \ldots p_n$;
Begin

While $(c_1c_2\ldots c_n > Peer[ind])$ **Do**
  $ind++$;
End While
If $(Peer[ind] == c_1c_2\ldots c_n)$ **Then**
  Return $Peer[ind]$;
Else
  If $(Peer[ind-1] == c_1c_2\ldots c_n)$ **Then**
    Return $Peer[ind-1]$;
  Else
    Return $c_1c_2\ldots c_n$ does not exists;
  End If
End If
End

The principle of this algorithm is to determine to which destination peer the source peer launches its request for a given key $c_1c_2\ldots c_n$, initially, towards the first peer higher than the key $c_1c_2\ldots c_n$, if it successful (*the destination peer holds the key* $c_1c_2\ldots c_n$), the request stopped. If not, it continues the research towards the predecessor peer. So once again, the request did not succeed, an information message indicating that the key $c_1c_2\ldots c_n$ does not exist in the network. For more illustration, we give the following example:

Example. When a given peer is looking for a resource with identifier k_{1420}, it initially locates the identifier peer which holds this key, all peers knowing that this one is in possession of the first peer higher than peer 1420, being the peer 1423. If the lookup algorithm do not found the desired key at this peer, it identifies a new destination peer, which is the predecessor of the peer 1423 whose numerical identification is lower; it is the peer 1342 in this case. It is important to note, that this algorithm has a low cost, because it runs locally on the level of each peer source, which seeks to locate a key $c_1c_2\ldots c_n$ without any overload on the network, especially in terms of sending and receiving messages (*overhead*). After having located the destination peer on which the key (*service*) sought is, a lookup algorithm with a minimum number of hops will be evoked.

3.3. Maintenance processes

Node joining: when a new node $u = u_1u_2\ldots u_n$ needs to join the proposed architecture (p_n), it follows the following steps:

1. Identify the insertion ring where the nth value of their identifiers is the same as u_n,
2. Searching in the ring identified above, the first node higher than the node to insert $u_1u_2\ldots u_n$, let be $v_1v_2\ldots v_n$ the founded node,
3. Insert the node such as $u_1u_2\ldots u_n$ is the node predecessor of the node $v_1v_2\ldots v_n$,
4. Update the links with the nodes $u^n, u^{n-1}, u^{n-2}, \ldots, u^2$ of the network to build the routing table,
5. Re-initialized the allocation of the resources of both peers successor and predecessor of the node to be inserted.

Node leaving: when a node $u = u_1u_2\ldots u_n$ needs to leave the network, or the ring where it appears, it launches the process of attribution of the resources, which it again holds between its peer successor and predecessor, according to the algorithm of attribution described previously (Algorithm 2). Then, it should be informed their linked nodes, in order to update their finger's table by the successor or predecessor of them. Finally, it leaves the network followed by the elimination of all the links with the nodes $u^n, u^{n-1}, u^{n-2}, \ldots, u^2$ of the network.

3.4. Lookup process

Lookup is started according to the numerical value of the identification of the source node and destination node, then, if they appear in the same ring, the appropriate Lookup algorithm in this case is executed, so that the source s reaches the destination d with a minimum of number hops, by looking for a path that enables us to materialize this constraint, which is sometimes founded by the successors of s or of course by their predecessors. If the both nodes (*source and destination*) appear in two different rings, another type of lookup appropriate to this new case will be activated. This lookup consists of identifying a node which appears in the same ring, where the node source or destination is figured. If this required node appeared in the ring where the node source is located, then it is viewed as a new destination to reach. It is considered as being a new source node if it appeared in the ring of the destination node.

A node sought from the node source or the node destination has a number of hops mostly equal to two, in the case where $s_n \neq d_n$, and only one hops, in the case where $s_n \neq d_n$ and $s_1 = d_n$, or $s_n \neq d_n$ and $s_n = d_1$ and intuitively equal to zero hops in the case of $s_n = d_n$. The first primary goal of our lookup algorithm is to determine the ring, where the source s and the destination d belong with optimized number of hops; at most it is equal to two. After making the nodes s and d in the same ring, a convergence process is launched for enabling us to deduce the new nodes s' and d' from the routing tables of s and d, in order to allow the node s to reach the node d, with a minimum number of hops.

In all previous cases, if the destination node is awaited by the node source (*lookup request is successful*), the appropriate lookup algorithm for each cases returns the optimized path, to reach the node sought starting from the source node. Otherwise, the lookup fails, if the destination node does not exist, or it does not hold the desired service (*the resource*).

3.4.1. Routing table description

The routing table of each node participating in the underlying architecture presented previously is defined according to Table 3. Moreover, to the n entries which are defined in the Pancake graph, we added $n-3$ more entries, these new entries introduced to this new routing table, in order to create a certain situation of convergence between the various nodes that desire to communicate.

3.4.2. Description of the lookup algorithm

A new lookup mechanism for an optimized path between any two arbitrary peers (*a source peer and a destination peer*), with a minimum number of hops according to the proposed architecture is proposed. The proposed source-destination algorithm follows four different cases:

- The nth value of the source and the destination are equal (i.e. $s_n = d_n$) s and d belong to the same ring,
- The nth value of the source and destination are different, as well as the beginning of the source and the nth value of the destination are equal (i.e. $s_n \neq d_n$ *and* $s_1 = d_n$),

Table 3
Routing table (*Finger Table*) of nodes

Entry number	The link
1	s^1
2	s^2
3	s^3
4	2^k
...	...
$n-1$	$s^{(n-1)}$
n	s^n
$n+1$	$s^3(s^2)$
...	...
$n+(n-3)$	$s^{n-1}[s^{n-2}(s^{n-3}(\dots(s^2)))]$
$n+(n-3)+1$	node successor
$n+(n-3)+2$	node predecessor

- The nth value of the source and destination are also different, as well as the nth value of the source and the beginning of the destination is equal (i.e. $s_n \neq d_n$ and $s_n = d_1$),
- The nth value of the source and the destination are different, (i.e. $s_n \neq d_n$).

In the last three cases, s and d belong to two different rings. In what follows, we will present in detail the four cases. Let $s = s_1 s_2 \ldots s_n$, $d = d_1 d_2 \ldots d_n$ the source and the destination nodes respectively, such that this algorithm is composed of four cases below:

- Case 1: $s_n = d_n$.
- Case 2: $s_n \neq d_n$ and $s_1 = d_n$.
- Case 3: $s_n \neq d_n$ and $s_n = d_1$.
- Case 4: $s_n \neq d_n$.

In the following, we describe and discuss different algorithms according to different cases.

- **Description and pseudo code of the Algorithm: Path Sour-Dest()**

Algorithm 4 (Path Sour-Dest()).

Procedure Path ($s = s_1 s_2 \ldots s_n, d = d_1 d_2 \ldots d_n$**)**
Begin
If ($s_n = d_n$) **Then**
  Convergence-Ring ($s = s_1 s_2 \ldots s_n, d = d_1 d_2 \ldots d_n$);
Else
  If ($s_1 = d_n$) **Then**
    $s' := s^n$;
    Select edge (s, s');
    Convergence-Ring ($s' = s'_1 s'_2 \ldots s'_n, d = d_1 d_2 \ldots d_n$);
  Else
    If ($s_n = d_1$) **Then**
      $d' := d^n$;
      Select edge (d', d);
      Convergence-Ring ($s = s_1 s_2 \ldots s_n, d' = d'_1 d'_2 \ldots d'_n$);
    Else
      For $i := n - 1$ to 2 step -1 **Do**
        $s' := s^i$;
        $s'' := s^{(i,n)}$;
        If ($s''_n = d_n$) **Then**
          Select edge (s, s');
          Select edge (s', s'');
          Convergence-Ring ($s'' = s''_1 s''_2 \ldots s''_n, d = d_1 d_2 \ldots d_n$);
        End If
      End For
    End If
  End If
End If
End

The first step of our proposed lookup algorithm is based on the idea that if the nodes source and destination are on the same ring, according to the last cipher's composing their identifiers, then we execute the process of convergence directly (*try to find the links which converge the source and destination nodes*). If not, we seek to reproduce them on the same ring before continuing our research.

- **Description of the pseudo code of the Convergence-Ring () Algorithm**

Algorithm 5 (Convergence-Ring ())**.**

```
Procedure Convergence-Ring (s = s1s2...sn, d = d1d2...dn)
s', d', min, x, y: Integer; min := s − d; x = s^2; y = d^2;
Begin
For i := n − 1 to 1 step −1 Do
   For j := n − 1 to 1 step −1 Do
     If (Dist − Min(s^i, d^j) < min) Then
        min := Dist − Min(s^i, d^j); s' := s^i; d' := d^j;
     End If
   End For
End For
d = y;
For i := n − 1 to 1 step −1 Do
   For j := 3 to n − 1 step 1 Do
     If (Dist − Min(s^i, d^j) < min) Then
        min := Dist − Min(s^i, d^j); s' := s^i; d' := d^j;
     End If
        d := d^j;
   End For
        d := y;
End For
d = y^2; s = x;
For i := n − 1 to 1 step −1 Do
   For j := 3 to n − 1 step 1 Do
     If (Dist − Min(s^j, d^i) < min) Then
        min := Dist − Min(s^j, d^i); s' := s^j; d' := d^i;
     End If
        s := s^j;
   End For
        s := x;
End For
s = x; d = y;
For i := 3 to n − 1 step 1 Do
   For j := 3 to n − 1 step 1 Do
     If (Dist − Min(s^i, d^j) < min) Then
        min := Dist − Min(s^i, d^j); s' := s^i; d' := d^j;
     End If
        d := d^j;
   End For
        s := s^i; d := y;
End For
Return (s', d', min);
End
```

For more explanation, we give the signification of variables used in this algorithm at Table 4.

This procedure is executed only, if the node source and the node destination are on the same ring. Their role is try to find other peers, enable to the peer source to reach destination peer, using the principle of the minimal distance

Table 4

Variables vs significations

Variable	Signification
min	The minimal distance between two arbitrary nodes according to their identifiers.
$Dist - Min$	The minimal distance between two arbitrary nodes according to node identifiers of their routing tables.

between each two peers that desire communicate, after comparisons of all possible permutations which are defined according to their routing tables, starting from the peer source towards the peer destination. Then, it enables us to choose the best possible permutation of the peer source and destination (*the best permutation is defined by the minimal distance between the bonds of their routing tables*). If the process of convergence obtains a new identifier peer which is a permutation of the peer source, then it is viewed as a new peer source, with the preservation of the bond which carries out us towards the old source. If also, they find new identifier peer which is a permutation of the destination peer, we look it as new peer destination; with preservation of the link leads to the old destination. It can also lead us to a permutation of the peer source and destination. In this case, we must preserve the two links that lead us to the old source and destination peer. It may be also, that these suggested two peers constitute the best solution of minimization between them. In this case, no bonds are preserved.

This process is repeated by the two new obtained nodes, as long as there is a new minimal distance according to their new routing tables.

After the execution of this process, which brings us to a better minimization of the number of hops between each two nodes, the following situations will be invoked.

- The situation where the peer source is the successor or the predecessor of the destination peer, in this case, the procedure continues its execution by a call to the function *Path Succ-Pred()*, which carries out towards the precision of the remainder and the required final path,
- A common pair, which is shared by the both peers source and destination, in this case, lookup is finished. It remains to deduce the path according to the bonds preserved previously,
- The situation where the peer source and destination share a common part, between the first ciphers which constitute their identifiers. If this degree of sharing is important, (i.e. *achieve a certain factor compared to the number of ciphers which constitute the identifiers of implemented network*), then, the research continues according to a technique which favorites the sharing. If not, we launch another technique which is based on the use of the identifiers peers resulting from the execution of the procedure *Path Sour-Dest()*.

These techniques are specified and detailed by Algorithm 6.

- **Pseudo code of the Algorithm: Closer-Convergence()**

Algorithm 6 (Closer-Convergence()).

Procedure Closer-Convergence ($s = s_1 s_2 \ldots s_n, d = d_1 d_2 \ldots d_n$)
Begin
If ($\exists$ common parts $\geqslant n/2$) **Then**
$s := s_{n-1} s_{n-2} \ldots s_2 s_1 s_n$;
$d := d_{n-1} d_{n-2} \ldots d_2 d_1 d_n$;
Else
s, d: takes the values when the first call of *Convergence-Ring()*;
End If
For $i := n - 1$ to 1 step -1 **Do**
  If ($s_i \neq d_i$) **Then**
    If ($s_1 = d_i$) **Then**
      Select edge (s, s^i); $s := s^i$;
    Else

If $(d_1 = s_i)$ **Then**
Select edge (d^i, d); $d := d^i$;
Else
Find k thats $s_k = d_i$;
Select edge $(s, s^{(k)})$;
Select edge $(s^{(k)}, s^{(k,i)})$;
$s := s^{(k,i)}$;
End If
End If
End If
End For
End

- **Description and pseudo code of the function Path Succ-Pred()**

The function *Path Succ-Pred()* constitutes the last phase of our source destination lookup algorithm. It enables to reach the requested destination, through the successors or the predecessors of a source node around the ring, in such manner to optimize the number of required hops. This function turns over the remainder of the path in order to reach the desired node. For more illustration, we give the signification of variables used in this algorithm in Table 5.

Algorithm 7 (Path Succ-Pred())**.**

Function Path Succ-Pred $(s = s_1 s_2 \dots s_n, d = d_1 d_2 \dots d_n)$
Succ, *Pred*, *Max*, *Min* :integer's;
$Max := Maximum\text{-}Ring(s_1 s_2 \dots s_{n-1} s_n)$;
$Min := Max^{n-1}$;
Begin
If $(s < d)$ **Then**
$Succ := d - s$;
$Pred := (Max - d) + (s - Min)$;
Else
$Succ := (Max - s) + (d - Min)$;
$Pred := s - d$;
End If
If $(Succ < Pred)$ **Then**
Locate the destination d by the successors of s;
Else
Locate the destination d by the predecessors of s;
End If
Return Path of s towards d;
End

Table 5
Variables vs significations

Variable	Signification
Min	The minimal value of identifier nodes in a given ring
Max	The maximal value of identifier nodes in a given ring
Succ	The successor of a given node
Pred	The predecessor of a given node

- **Description and pseudo code of the function Maximum-Ring()**

This function returns the maximum identifier in term of numerical value, among the identifiers of the nodes which can belong to a given ring. This maximum numerical value is used by the lookup algorithm (*in the four cases*) in order to already attaining the requested node with minimal number of hops.

Algorithm 8 (Maximum-Ring()).

```
Function Maximum-Ring (s1 s2 ... sn−1 sn)
Temp, i, j :integers;
Begin
For i := n − 1 to 2 step −1 Do
    For j := 1 to i − 1 step −1 Do
        If (si > sj) Then
            Temp := si;
            si := sj;
            sj := Temp;
        End If
    End For
End For
Return (s1 s2 ... sn−1 sn);
End
```

- **Description and pseudo code of the function Distance-Minimal()**

This function returns the minimal distance between each two arbitrary peers of the network belonging to the same ring. It receives the numerical identification of these two peers as arguments. This minimal distance is used in the choice of the peers which enable to minimize and to converge these two peers, from a permutation of the peer source and/or peer destination. It is called by the procedure *Convergence-Ring()* in order to achieve the desired node with minimal number of hops. We use the same denotation as in Algorithm 7.

Algorithm 9 (Distance-Minimal()).

```
Function Dist-Min (s, d)
Succ, Pred, Max, Min, Dist :integer's;
Max := Maximum-Ring(s1 s2 ... sn−1 sn);
Min := Max^(n−1);
Begin
If (s < d) Then
    Succ := d − s;
    Pred := (Max − d) + (s − Min);
Else
    Succ := (Max − s) + (d − Min);
    Pred := s − d;
End If
If (Succ < Pred) Then
    Dist := Succ;
Else
    Dist := Pred;
End If
Return Dist;
End
```

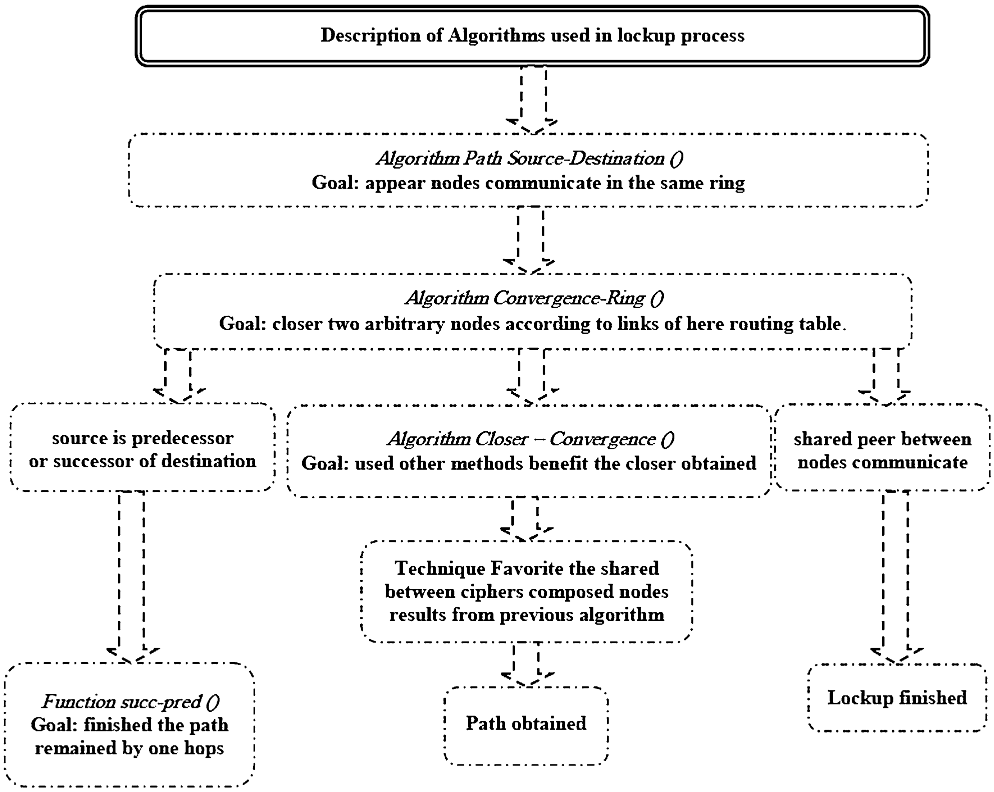

Fig. 4. Algorithms of the proposed architecture.

- **Flow chart of the proposed lookup process**

For more illustration, we give the a flow chart (see Fig. 4) about different procedures of our proposed architecture used in each step, follow-up by their principal goals.

3.5. *Course of the proposed technique*

To illustrate some details of our lookup algorithm, we give the examples of unfolding in a graph 5-pancakes (p_5) for the Case 2, Case 3 and one example in a graph 7-pancakes (p_7) for case 4.

- **Unfolding for the second case: where** $s_n \neq d_n$ **and** $s_1 = d_n$
  Let $s = 13524$ and $d = 43521$ two peers (*source and destination respectively*), the lookup of the peer d starting from the peer s is carried out as follows:
  - **The execution of the procedure *Path Sour-Dest*(**$s = 13524$**,** $d = 43521$**), generates the following instructions**:
    - Select edge $(s, s') \Rightarrow (13524, 42531)$.
    - A call to the procedure *Convergence-Ring*($s' = 42531$, $d = 43521$).
  - **The execution of the procedure *Convergence-Ring*(**$s = 42531$**,** $d = 43521$**), generates the following instructions**:
    - According to the bonds of the routing table of s which are: 42531, 24531, 52431, 35241, 54231, 32451.

- ∗ And according to the bonds of the routing table of d which are: 43521, 34521, 53421, 25341, 54321, 23451.
- ∗ It is concluded that the two nodes which constitute the minimal distance are: $s' = 35241$ and $d' = 34521$.
- ∗ The node $d' = 34521$ is the node predecessor of the $s' = 35241$.
- ∗ This situation is charged by the function *Path Succ-Pred*($s = 35241$, $d = 34521$), which we allow to complete the remainder of the path.
- ∗ Therefore, the final path from $s = 13524$ to $d = 43521$ is: 13524 ⇒ 42531⇒ 35241 ⇒ 34521 ⇒ 43521 – 4 hops –

- **Unfolding for the third case: where $s_n \neq d_n$ and $s_n = d_1$**
  Let $s = 54132$ and $d = 25314$ two peers (*source and destination respectively*), the lookup of the peer d starting from the peer s is carried out as follows:

  ∗ **The execution of the procedure *Path Sour-Dest*($s = 54132$, $d = 25314$), generates the following instructions**:

  - ∗ Select edge(d', d) ⇒ (41352, 25314).
  - ∗ A call to the procedure Convergence-Ring(s = 54132, d' = 41352).

  ∗ **The execution of the procedure *Convergence-Ring*($s = 54132$, $d = 41352$), generates the following instructions**:

  - ∗ According to the bonds of the routing table of s which are: 54132, 45132, 14532, 31452, 15432, 34512.
  - ∗ And according to the bonds of the routing table of d which are: 41352, 14352, 31452, 53142, 34152, 51432.
  - ∗ It is deduced that the two nodes s and d share a common node which is: 31452.
  - ∗ In this situation, lookup is finished; therefore the final path from $s = 54132$ to $d = 25314$ is: 54132 ⇒ 31452 ⇒ 41352 ⇒ 25314 – 3 hops –

- **Unfolding for the fourth case: where $s_n \neq d_n$**:
  Let s = 5472163 and d = 5726134 two peers (*source and destination respectively*), the lookup of the peer d starting from the peer s is carried out as follows:

  ∗ **The execution of the procedure *Path Sour-Dest*($s = 5472163$, $d = 5726134$), generates the following instructions**:

  - ∗ Select edge (s, s') ⇒ (5472163, 4572163).
  - ∗ select edge (s', s'') ⇒ (4572163, 3612754).
  - ∗ A call to the procedure Convergence-Ring($s'' = 3612754$, $d = 5726134$).

  ∗ **The execution of the procedure *Convergence-Ring*($s = 3612754$, $d = 5726134$), generates the following instructions**:

  - ∗ According to the bonds of the routing table of s which are: 3612754, 6312754, 1632754, 2163754, 7216354, 5721634, 1362754, 2631754, 7136254, 5263174.
  - ∗ And according to the bonds of the routing table of d which are: 5726134, 7526134, 2756134, 6275134, 1627534, 3162754, 2576134, 6752134, 1257634, 3675214.
  - ∗ It is deduced that the nodes which constitute the minimal distance are: $s' = 5721634$ and $d' = 5726134$.
  - ∗ In this situation, the procedure Convergence-Ring calls the procedure Closer-Convergence(s', d'), since the degree of sharing equal to $3 \geqslant 7/2$, therefore generated instructions by this one are: ($s = 5721634$, $d = 5726134$) ⇒ ($s' = 6127534$, $d' = 1627534$).
    $s' = s^2 \Rightarrow s' = 1627534 = d'$.
  - ∗ Therefore, the final path from $s = 5472163$ to $d = 5726134$ is: 5472163 ⇒ 4572163 ⇒ 3612754 ⇒ 5721634 ⇒ 6127534 ⇒ 1627534 ⇒ 5726134 – 6 hops –

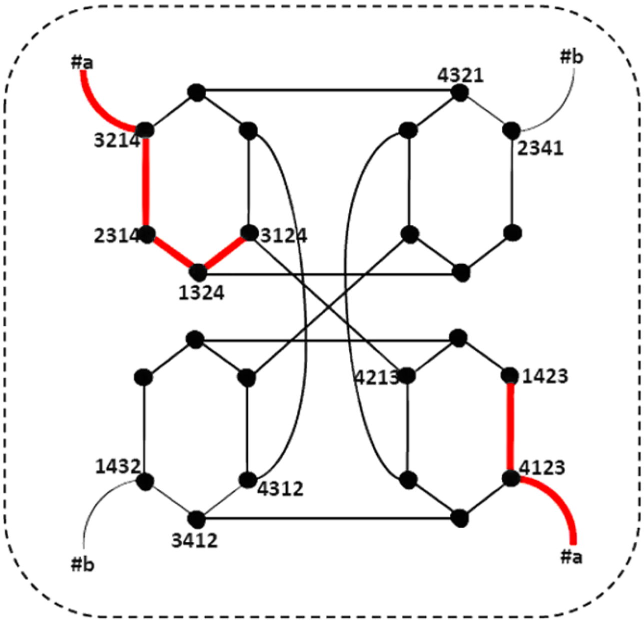

Fig. 5. Example of routing between peers 1423 and 3124 in P_4.

3.6. *Comparison with the Route routing algorithm in Pancake graph*

The most important aspect of the proposed lookup algorithm is the improvement of the routing in term of number of hops between any two arbitrary nodes. Indeed, if we take an example of two peers $s = 1423$ and $d = 3124$, then the algorithm *Route ()* finds the path: 1423 $\Rightarrow$ 4123 $\Rightarrow$ 3214 $\Rightarrow$ 2314 $\Rightarrow$ 1324 $\Rightarrow$ 3124 (see Fig. 5). Therefore, a path length is equal to 5 hops. However, the proposed algorithm finds a path length equal to 3 hops only, it is 1423 $\Rightarrow$ 2413 $\Rightarrow$ 4213 $\Rightarrow$ 3124 (see Fig. 6). Such a factor is very important.

3.7. *Advantages and shortcoming of the proposed solution*

- **Advantages**
  - Our lookup algorithm gives paths with a minimum number of hops,
  - A great flexibility of addition and suppression of nodes,
  - Facilitate the manner of attribution and localization of the keys,
  - Provide a certain percentage of load balancing in term of number of resources holding each peer.
- **Shortcoming**
  - Increase the number of entries in the routing table compared to the number of entries in the classical graph of Pancake by $n - 3$ entries,
  - Do not provide a total distribution balanced in term of number of resources that each peer holds.

4. Performance evaluation

To improve the performance of our proposed solution in term of node to node type of routing (*unicast or point to point*) discussed inside our proposed overlay topology, where their identifier nodes uses the process of permutation

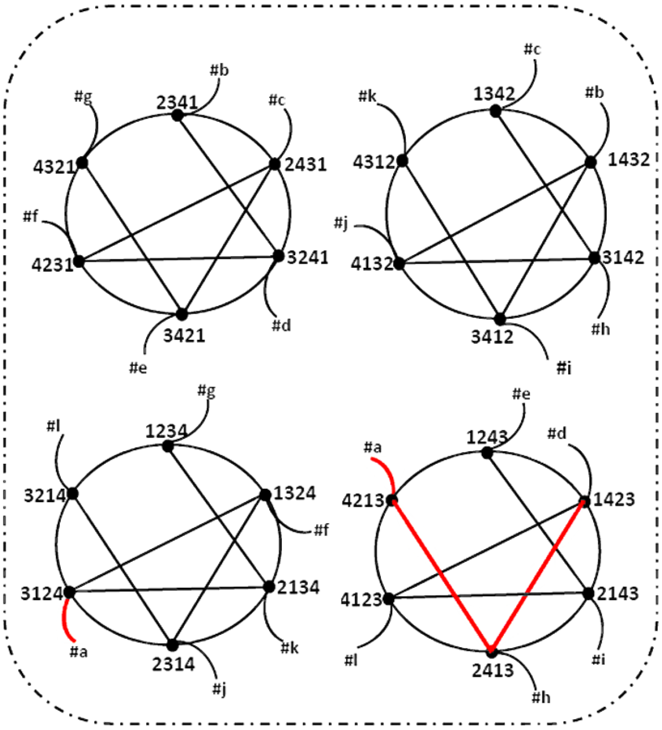

Fig. 6. Example of routing between peers 1423 and 3124 in P_4 proposed.

to connect all these nodes to each other, providing a requirement to search other solutions for the node to node routing problem that are based on the Pancake graphs.

Among the solutions founded is the Route algorithm based on classical Pancake graphs used to identify one path between a random pair of nodes, which used also by NS algorithm [23] in such cases to find disjoint paths between a node and several distinct nodes. Find disjoint paths is efficient in parallel systems, under a fault topology and not efficient when we transfer the same data to the group of destinations nodes.

In the other side, and as we have discuss above, our proposition is based on a ring topology (*inspired from Chord protocol*). So, set $(n-1)!$ identifier nodes that finished by the same ciphers around a ring in ascending way provides a really optimization in terms of the number of hops, between each both neighboring nodes which are never neighboring in an under Pancake P_{n-1}, as a result, $(n-1)! - 1$ both neighboring nodes can communicate only by 1 hop (*successor/predecessor node*). In addition, the same both nodes are connected by a number of links superior or equal than 2 hops, according to the topology of classical Pancake graphs. For example, in P_8, we have $(8-1)! - 1 = 5039$ both neighboring nodes that are connected by only 1 hop, according to our overlay topology. For source node 78654132 and destination node 78653412 we have:

- For our proposed solution: 78654132 ⇒ 78653412 then only 1 hop is needed.
- For the Route algorithm: 78654132 ⇒ 14568732 ⇒ 37865412 ⇒ 56873412 ⇒ 78653412 then 4 hops are needed.

For more validation purpose, we have carried a series of lookup tests aimed to bring up and evaluate the performance of the proposed method.

In order to measure the performance of the proposed lookup process and to highlight the importance of hops number, delay of paths founded, we compare our proposed lookup algorithm with routing algorithm proposed by Y. Suzuki and K. Kaneko for the same type of graph (*same number of nodes in the graph*) and between the same source node and destination node.

Our algorithms are implemented by a programming language JAVA (*Eclipse*), where the source node s is fixed. After, we select the destination node d other than s randomly, and invoked the proposed algorithm. Finally, we measure the number of hops of the obtained path and its delay.

The results obtained are based on three types of graphs: a graph p_6 (720 *nodes*), a p_7 graph (5040 *nodes*) and a graph p_8 (40320 *nodes*). Results are summarized and illustrated in the following figures.

4.1. Evaluation in term of number of hops

- **For graph p_6 (720 nodes)**

The curves illustrated in Fig. 7 give a graphic representation of the obtained results, for the various tests carried out according to the number of hops, to convey a request from a source node to a destination node on a Pancake graph p_6. Figure 7 gives the simulation results for different tests carried out on a graph p_6 containing 720 nodes. The curves show well that the routing path between two given peers is improved in terms of number of hops, for the proposed solution compared to the *Route()* algorithm.

- **For graph p_7 (5040 nodes)**

In Fig. 8, various tests carried out depending on the number of hops and even of source, destination on a graph p_7. According to the Fig. 8, we notice that the obtained results using the proposed lookup algorithm are optimized to a

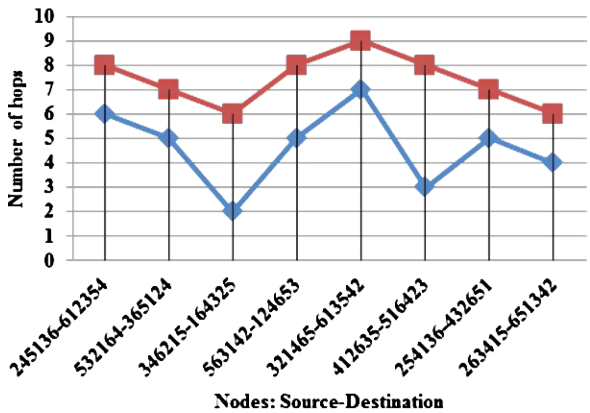

Fig. 7. Number of hops between two peers (*source-destination*) on a graph p_6.

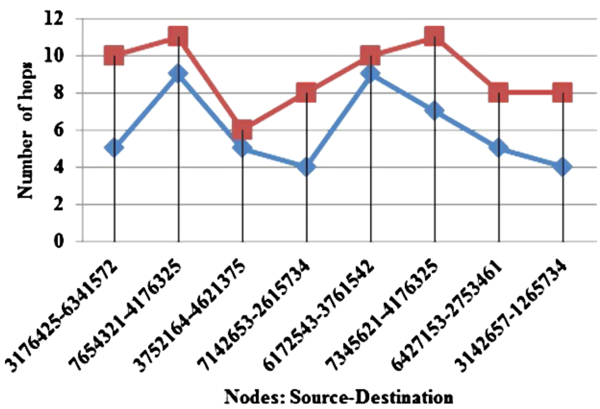

Fig. 8. Number of hops between two peers (*source-destination*) on a graph p_7.

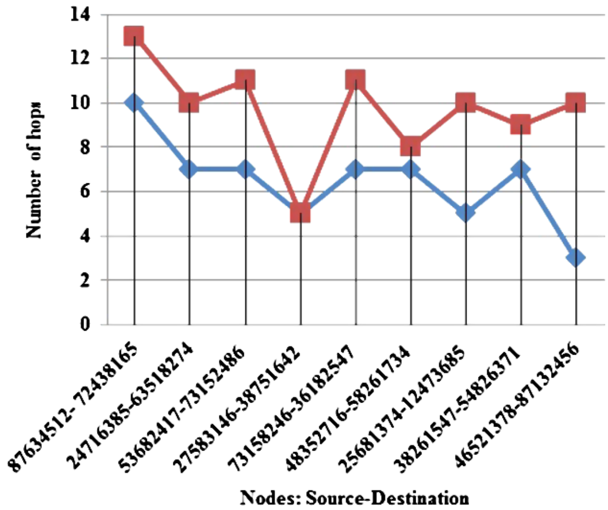

Fig. 9. Number of hops between two peers (*source-destination*) on a graph p_8.

minimum number of hops compared to the *Route()* algorithm employed in the classical graph of Pancake, through the various tests carried out on a graph p_7, which is composed of 5040 nodes.

- **For graph p_8 (40320 nodes)**

In Fig. 9, various tests carried out according to the number of hops and even of source, destination on a graph p_8.

A series of tests carried out on a graph p_8 composed of 40320 nodes. The proposed routing algorithm is compared to the *Route()* algorithm used in the classical graphs of Pancakes. The best test gives an improvement of seven hops on the one hand, and on the other hand there are cases where the number of hops is identical for both routing techniques. According to the two curves illustrated in Fig. 9, we can conclude that our proposed lookup algorithm outperforms the Route algorithm in terms of number hops.

4.2. Evaluation in term of delay

For performance evaluation, we consider also the delay as an important performance metric. So, to compare the obtained delay of the resulting paths from each type of lookup process (*lookup process of the proposed P2P architecture, lookup process of the Pancake graph*), we proceed as follows: to measure the delay of a path between arbitrary nodes, we should evaluate the delay of each hop composed this path. So, for a given hop between a node $X = x_1x_2 \dots x_m$ and a node $Y = y_1y_2 \dots y_m$, its delay is calculated as following:

- If $x_m = y_m$, i.e. the nodes X and Y are in the same ring or the same part of Pancake, its delay D takes the value of x_m or y_m,
- If $x_m \neq y_m$, i.e. the nodes X and Y are in different rings or different parts of Pancakes, its delay D takes the absolute value: $|x_m - y_m|$.

Example. To evaluate the delay of path obtained by the proposed lookup process, we consider a source node: 42531867 which looks for the destination node: 53874162 in a Pancake graph p_8.

42531867 $\overset{7}{\Rightarrow}$ 24531867 $\overset{|7-2|}{\Rightarrow}$ 76813542 $\overset{2}{\Rightarrow}$ 53186742 $\overset{2}{\Rightarrow}$ 78351642 $\overset{2}{\Rightarrow}$ 78354162 $\overset{2}{\Rightarrow}$ 53874162. Delay: $D =$ 20 ms.

- **For graph p_7 (5040 nodes)**

In Fig. 10, various tests carried out according to the delay obtained from each path of the proposed routing algorithm and *Route()* algorithm used in Pancake graph for any two arbitrary nodes: source and destination on a

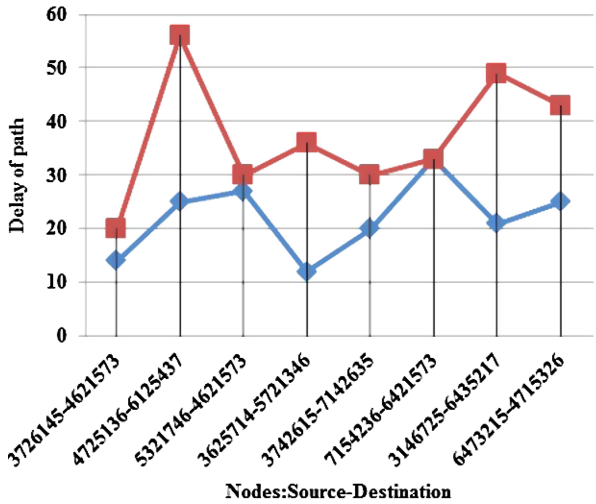

Fig. 10. Delay between two arbitrary peers (*source-destination*) on a graph p_7.

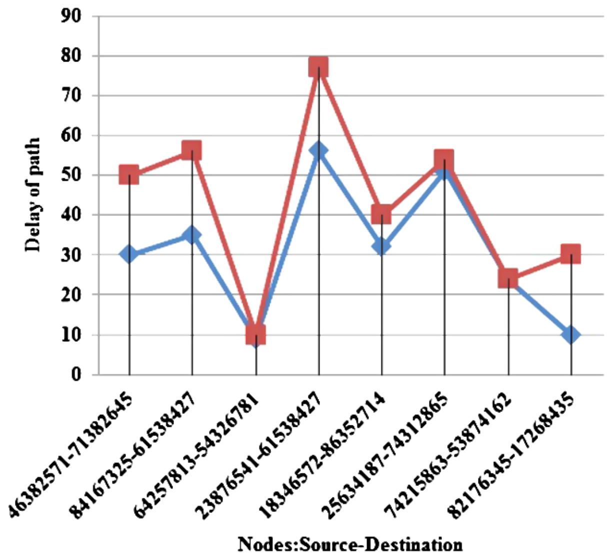

Fig. 11. Delay between two arbitrary peers (*source-destination*) on a graph p_8.

graph p_7. According to the two curves illustrated in Fig. 10, we show that the delay of the path given by our proposed lookup algorithm is less or equal to the delay of path given by the Route algorithm in a Pancake graph p_7 and in all tests.

- **For graph p_8 (40320 nodes)**

Figure 11 shows results of tests carried out in a Pancake graph p_8. p_8 is composed of 40320 nodes. The results show that our algorithm outperforms the Route algorithm. This aspect is due to the optimisation given by the local search in many phases of our algorithm.

5. Conclusion and future work

In one to one type of communication and real time applications, lookup acceleration is critical challenge. The acceleration is done in terms of number of hops from source node to destination node but also in terms of delay, particularly for P2P networking which implemented on application layer where the physical proximity is not well considered.

In this paper, we have exploited the propriety of permutation inspired from the graph of Pancake, in order to give a solution for the problem of lookup acceleration and optimization in P2P networks, by the minimization of the number hops and also the delay. For that, our proposed solution is based on two main axes.

The first one concerns the reference and the allowance of the resources, which are on the network by keys distributed on the whole of the peers, so that, either resource is attributed to the first peer higher in term of numerical value, only if it has less resources by report to its predecessor. Otherwise, it will be allotted to this predecessor, in order to provide a certain percentage of load balancing in the manner of resources distribution to the peers of the network.

The second axis concerns the lookup process which is based on the principle of permutations, which is considered as being the strong point of the Pancake graph, it allows a rapid convergence according to local treatment in some phases. The hierarchical rings architecture of our proposed model is inspired from Chord protocol, which allows an improvement of lookup in terms of the number of hops and the delay between to arbitrary nodes in the network.

The performance evaluation thought a comparison with the *Route()* algorithm in terms of number of hops and also delay using multiple types of Pancake graphs shows that our proposed architecture and particularly the lookup process is well improved.

In terms of perspectives, we envision to add:

- The fault-tolerance of the peers in the selected path (*peer failing*),
- Extend our architecture to support many to many type of group communication.

References

[1] S. Akers and B. Krishnamurthy, A group-theoretic model for symmetric interconnection networks, *IEEE, Transactions on Computing* (1989).

[2] M. Amad, D. Aïssani, A. Meddahi, M. Benkerrou and F. Amghar, De Bruijn Graph based solution for lookup acceleration and optimization in P2P networks, *Wireless Personal Communications* **85**(3) (2015), 1471–1486. doi:10.1007/s11277-015-2851-y.

[3] M. Amad, A. Meddahi and D. Aissani, P2P networks management survey, *International Journal of Computer Sciences Issues (IJCSI)* **9**(1) (2012), 139–148.

[4] S. Androutsellis-Theotokis and D. Spinellis, A survey of peer-to-peer content distribution technologies, *ACM Computing Surveys* **36**(4) (2004), 335–371. doi:10.1145/1041680.1041681.

[5] J. Aspnes and G. Shah, Skip graphs, In: *Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, Baltimore, MD, USA, 12–14 January 2003, pp. 384–393.

[6] M. Camelo, D. Papadimitriou, L. Fàbrega and P. Vilà, *Efficient Routing in Data Center with Underlying Cayley Graph*, Springer International Publishing, Switzerland, 2014.

[7] A. Deng and Y. Wu, De Bruijn digraphs and affine transformation, *Eur. J. Comb.* **26**(8) (2005), 1191–1206. doi:10.1016/j.ejc.2004.06.018.

[8] Y.-P. Deng and X.-D. Zhang, Small cycles in the Pancake graph, *ARS Mathematica Contemporanea* **7** (2014), 237–246.

[9] P. Desnoyers, D. Ganesan and P. Shenoy, TSAR: A two tier sensor storage architecture using interval skip graphs, in: *SenSys 05*, San Diego, California, USA, 2005.

[10] D. Eppstein, M.T. Goodrich and J.Z. Sun, The skip quadtree: A simple dynamic data structure for multidimensional data, in: *21st ACM Symp. on Computational Geometry (SCG)*, 2005.

[11] S. Femmam, M.F. Zerarka and M.I. Benakila, New Approach Construction for Wireless ZigBee Sensor Based on Embedding Pancake Graphs, *Network and Communication Technologies* **1**(1) (2012). Available at: http://www.ccsenet.org/journal/index.php/nct/article/view/17586. doi:10.5539/nct.v1n1p7.

[12] G. Fertin and A. Raspaud, A survey on knodel graphs, *Discrete Applied Mathematics* **137** (2004), 173–195. doi:10.1016/S0166-218X(03)00260-9.

[13] G. Fertin, A. Raspaud, O. Sykora, H. Schorder and I. Vrto, *Diameter of Knodel Graph, 26th International Workshop on Graph-Theoretic Concepts in Computer Science (WG2000)*, Lecture Notes in Computer Science, Vol. 1928, Springer-Verlag, 2000, pp. 149–160. doi:10.1007/3-540-40064-8_15.

[14] P. Fraigniaud and P. Gauron, D2B: A De Bruijn based content-addressable network, *Theor. Comput. Sci.* **355**(1) (2006), 65–79. doi:10.1016/j.tcs.2005.12.006.

[15] A.-T. Gai and L. Viennot, Broose: A practical distributed hash table based on the De-Bruijn topology, in: *Proceedings of the 4th International Conference on Peer-to-Peer Computing (P2P)*, Zurich, Suisse, 2004, pp. 167–174.

[16] M.T. Goodrich, M.J. Nelson and Z.J. Sun, The rainbow skip graph: A fault-tolerant constant-degree distributed data structure, in: *SODA'06: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, New York, NY, USA, ACM, 2006, pp. 384–393. doi:10.1145/1109557.1109601.

[17] A. Gupta and L.K. Awasthi, Peer-to-peer networks and computation: Current trends and future perspectives, *Computing and Informatics* **30** (2011), 559–594.

[18] H. Harutyunyan and J. He, A new peer-to-peer network, in: *PerCom 2007 Workshops. Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, 2007, pp. 120–125.

[19] N.J.A. Harvey, M.B. Jones, S. Saroiu, M. Theimer and A. Wolman, Skip net: A scalable overlay network with practical locality properties, in: *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems – Volume 4, USITS'03*, Berkeley, CA, USA, 2003, pp. 9.

[20] M.F. Kaashoek and D.R. Karger, Koorde: A simple degree-optimal distributed hash table, in: *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, LNCS, Vol. 2735, Springer, Berlin/Heidelberg, Berkeley, California, USA, 2003, pp. 98–107.

[21] M. Karpovsky, L. Levitin and M. Mustafa, Optimal turn prohibition for deadlock prevention in networks with regular topologies, *IEEE Transactions on Control of Network Systems* **1**(1) (2014), 74–85. doi:10.1109/TCNS.2014.2304869.

[22] J. Kathryn and J. Sullivan, The pancake problem: Improving the bounds for sorting by prefix permutations, Mathematics, Discipline, University of Minnesota, Morris, April 2005.

[23] K. Kaneko and Y. Suzuki, Node-to-set disjoint paths problem in pancake graphs, in: *IEICE TRANS*, 2003.

[24] A. Lars, D. Eppstein and M.T. Goodrich, Skip-webs: Efficient distributed data structures for multi-dimensional data sets, in: *ACM SIGAT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, ACM Press, New York, 2005, pp. 69–76.

[25] C. Lavault, *Embeddings into the Pancake Interconnection Network*, LIPN, CNRS ESA 7030, University Paris 13 Villetaneuse, France, April 1999.

[26] D. Loguinov, J. Casas and X. Wang, Graph-theoretic analysis of structured peer-to-peer systems: Routing distances and fault resilience, *IEEE/ACM Trans. Network.* **13**(5) (2005), 1107–1120. doi:10.1109/TNET.2005.857072.

[27] P. Maymounkov and D. Mazières, Kademlia: A peer-to-peer information system based on the xor metric, in: *Proceedings of 1st International Workshop on Peer to Peer Systems (IPTPS'02)*, Cambridge MA, USA, 2002, pp. 53–65. doi:10.1007/3-540-45748-8_5.

[28] W. Pugh, Skip lists: A probabilistic alternative to balanced trees, *Communications of the ACM* **33**(6) (1990), 668–676. doi:10.1145/78973.78977.

[29] Y. Qifeng, X. Tianyin, Y. Baoliu, L. Sanglu and C. Daoxu, Skip stream: A clustered skip graph based on-demand streaming scheme over ubiquitous environments, in: *IEEE, International Conference on Parallel Processing*, Vienna, Austria, 2009.

[30] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker, A scalable content-addressable network, in: *IN Proc. ACM SIGCOMM*, 2001, pp. 161–172.

[31] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek and H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for Internet applications, in: *ACM SIGCOMM'01*, San Diego, California, USA, 2001, pp. 149–160.

[32] Y. Suzuki and K. Keiichi, An algorithm for node-disjoint paths in Pancake graphs, *IEICE TRANS, INF. Syst.* **E86-D**(3) (2003), 610–615.