Variations of Chord lookup algorithm in P2P networks

Project Proposal
Team Gulties
Distributed Systems Team Research Investigation

Team Gulties - Members

Karteek Mekala (kkm6815@cs.rit.edu)

Ragdeep Moturu (rxm3575@cs.rit.edu)

Sowmya Yarlagadda (sxy5199@cs.rit.edu)

Agenda

- Recap
- Hypothesis
- Analysis of Paper 1
- Analysis of Paper 2
- Analysis of Paper 3
- Software Design
- Next Steps
- Questions

Recap

Lookup routing protocols for peer-to-peer networks.

Popular protocols include CAN, Chord, Pastry and Tapestry

We are studying Chord and two of its improvements: B-Chord and GA-Chord

Hypothesis

The average path length of the Chord protocol can be reduced by making using of group autonomy and bi-directional lookup algorithms.

Analysis of Paper 1

Title

Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications

Authors

Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan

Publication Year: 2003

Journal: IEEE/ACM Transactions on Networks

Page numbers: 17 - 32

URL: http://ieeexplore.ieee.org/search/srchabstract.jsp?tp=&arnumber=1180543

Chord Motivation

Chord addresses the following requirements for a peer-to-peer lookup protocol.

- Load Balance
- Decentralization
- Scalability
- Availability
- Flexible Naming

Chord - Basics

- Consistent Hashing

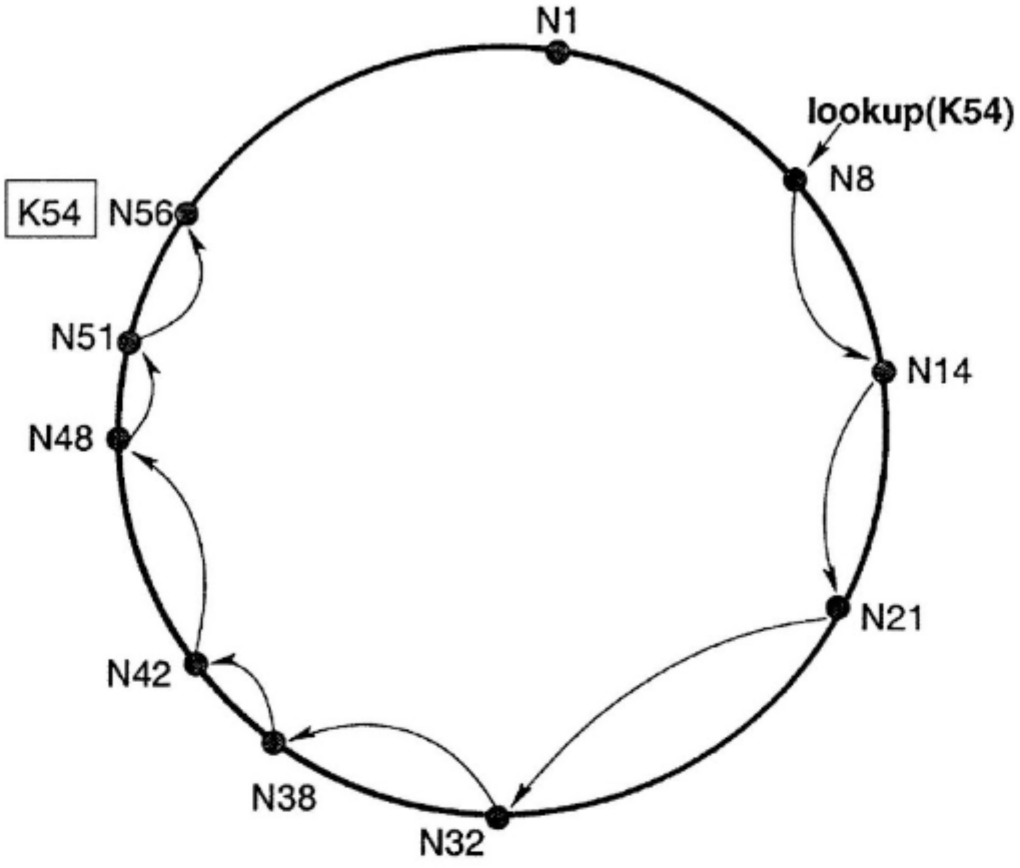
Identifiers are ordered on an identifier circle modulo 2^m. Using consistent hashing, key 'k' is assigned to the first node whose identifier is equal to or follows k in the identifier space.

- Simple Key location

Each node needs only to know how to contact its current successor node on the identifier circle

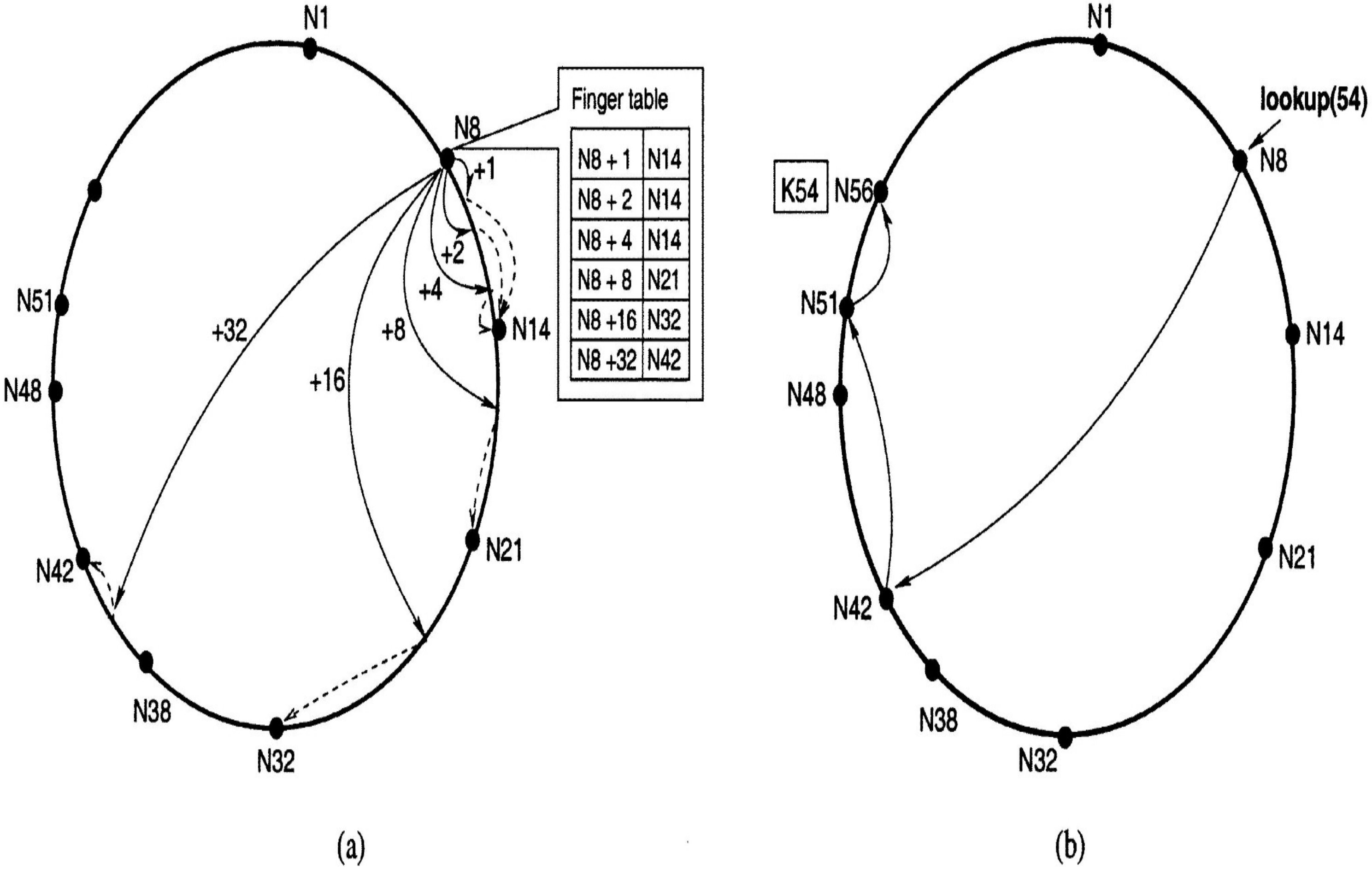
Chord - Simple Key Location

```
// ask node n to find the successor of id
n.find_successor(id)
        if(id ∈ （n, successor]）
                return successor;
        else
                // forward the query around the circle
                return successor.find_successor(id);
```



(b)

Chord Lookup example



Finger table

N8 + 1	N14
N8 + 2	N14
N8 + 4	N14
N8 + 8	N21
N8 +16	N32
N8 +32	N42

(a)

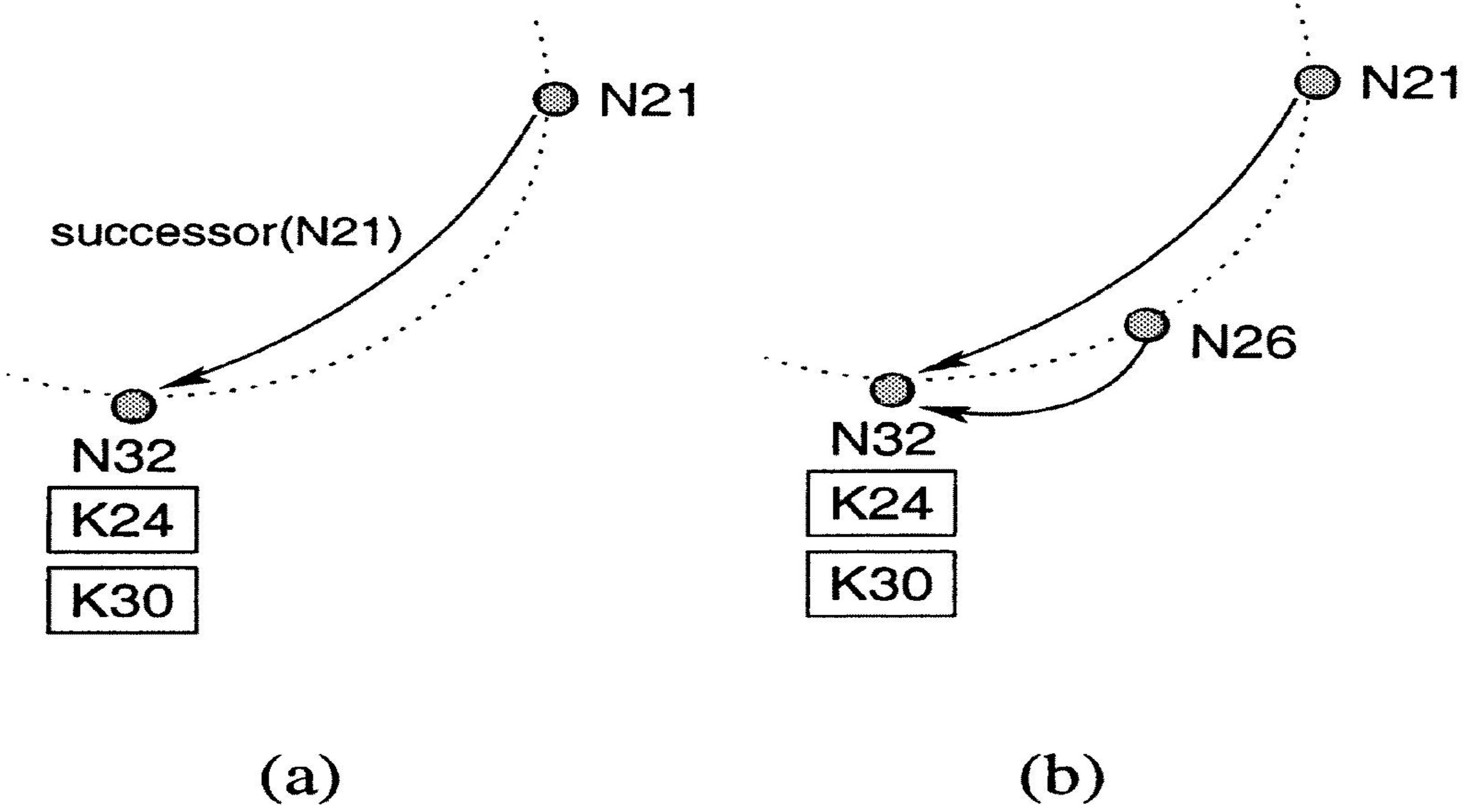
(b)

Chord-Look up Protocol

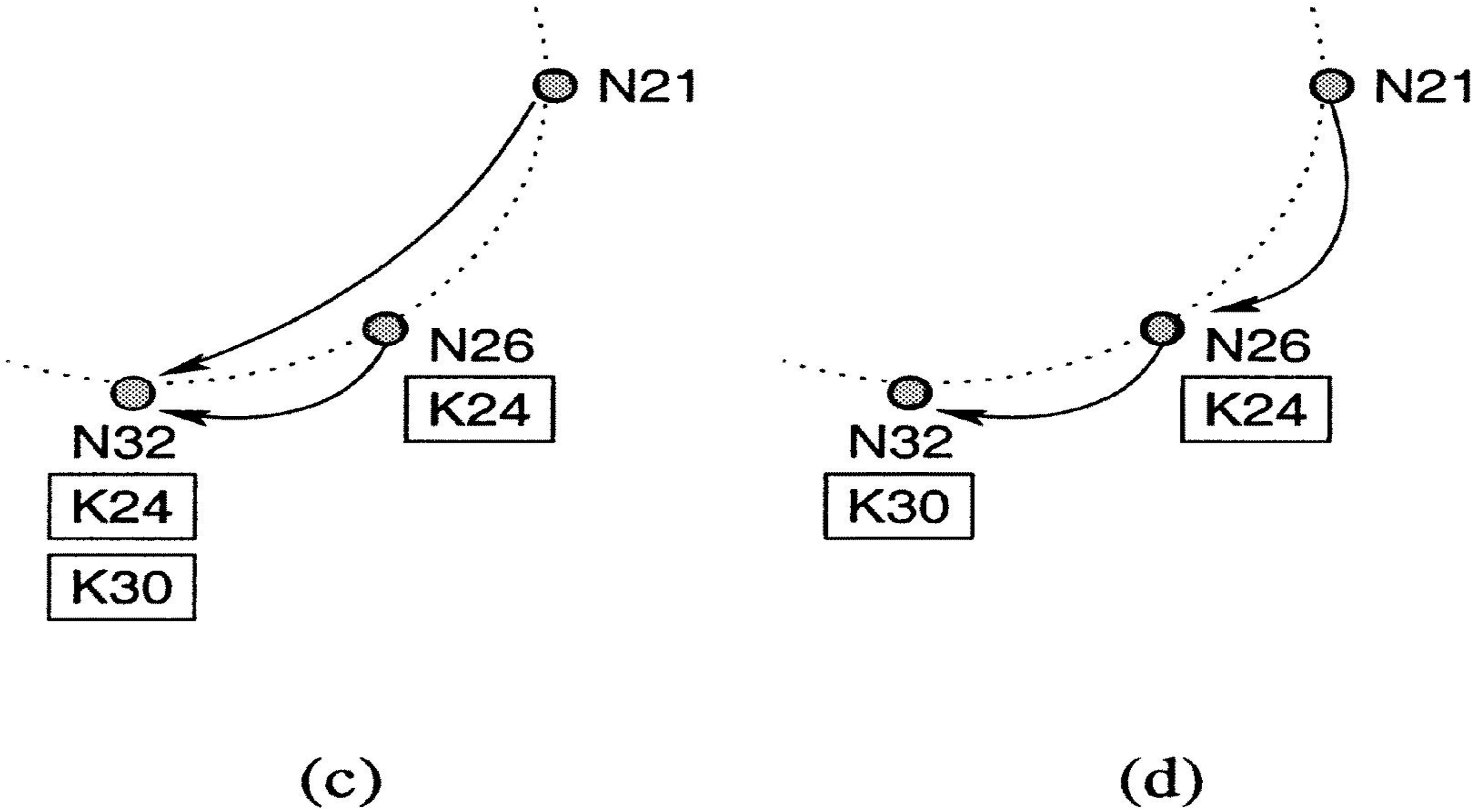
```
// ask node n to find the successor of id
n.find_successor(id)
    if( id $\in (n, successor]$)
        return successor;
    else
        n' = closest_preceding_node(id);
        return n'.find_successor(id);

// search the local table for the highest predecessor of id
n.closest_preceding_node(id)
    for i = m downto 1
        if( finger[i] $\in (n, id)$ )
            return finger[i];
    return n
```

Chord-Node joining Example



Chord-Node Joining continued...



Pseudo code of joining/leaving nodes ...

```
// create a new Chord ring
n.create()
        predecessor = nil;
        successor = n;

// join a Chord ring containing node n'
        n.join(n')
        predecessor = nil;
        successor = n'.find_successor(n);

// called periodically, verifies n's immediate successor
// and tells the successor about n
n.stabilize()
        x = successor.predecessor;
        if( x ∈ (n, successor) )
                successor = x;
        successor.notify(n);
```

Pseudo code of joining/leaving nodes

```
// n' thinks it might be our predecessor
n.notify(n')
    if( predecessor is nil or n' belongsTo (predecessor, n))
        predecessor = n';

// called periodically, refreshes finger table entries
// next stores the index of the next finger to fix
n.fix_fingers()
    next = next +1;
    if( next > m )
        next = 1;
    finger[next] = find_successsor( n + 2^next - 1 );

// called periodically, checks whether predecessor has failed
n.check_predecessor()
    if( predecessor has failed )
        predecessor = nil;
```

Chord Results

High probability that the length of the path to resolve a query is O(logN).

When an Nth node joins or leaves the network only an O(1/N) fraction of the keys are moved to different location.

Analysis of Paper 2

Title

B-Chord: Bi-directional Routing DHT based on Chord

Authors

Hongwei Chen, Zhiwei ye

Publication Year: 2008

Journal: 12th International Conference on Computer Supported Cooperative Work in Design, 2008.

Page numbers: 410 - 415

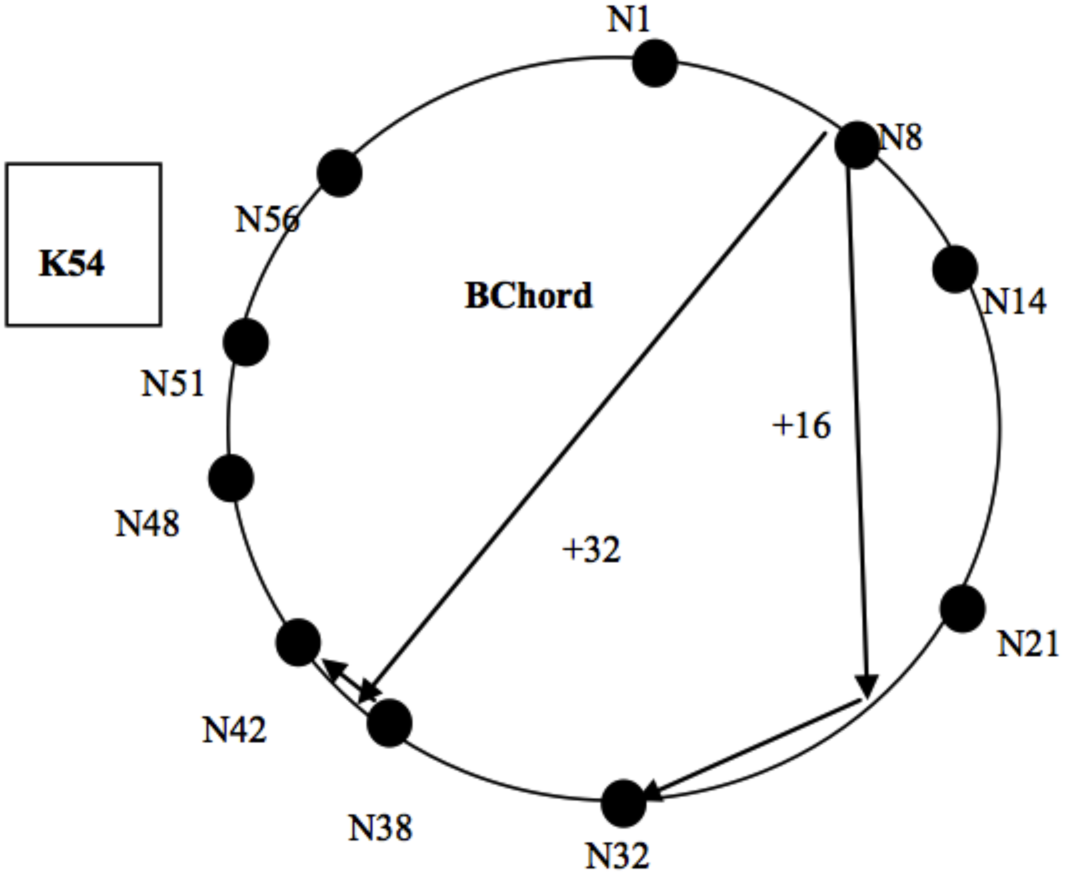
URL: http://ieeexplore.ieee.org/search/srchabstract.jsp?tp=&arnumber=4537014

B-chord Overview

- Query in clockwise and anticlockwise direction

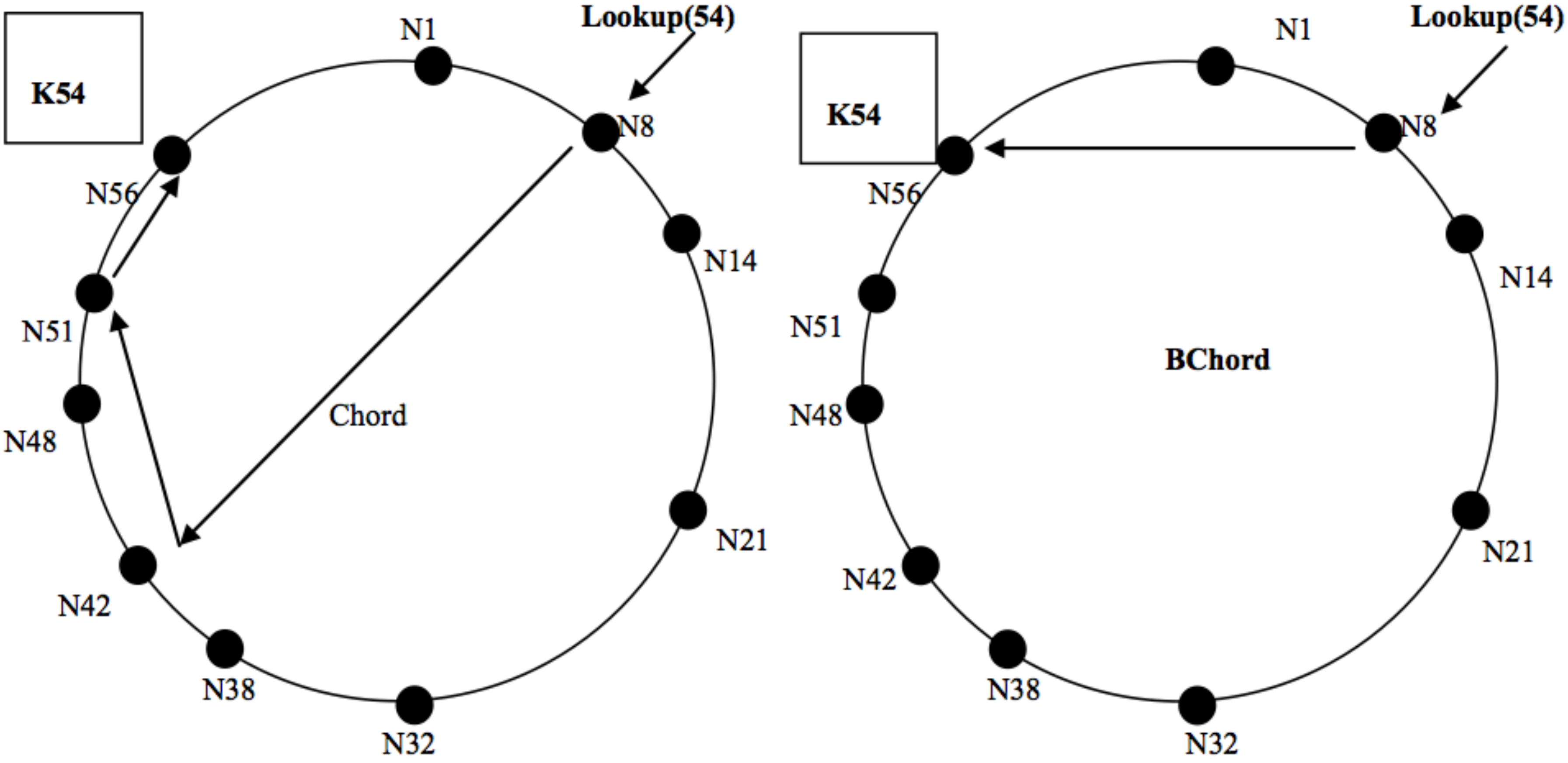
- Finger table entries for nodes in both directions, therefore double the number of entries

B-Chord Routing Example - Finger Tables



N8 + 1	N14
N8 - 1	-
N8 + 2	N14
N8 - 2	-
N8 + 4	N14
N8 - 4	-
N8 + 8	N21
N8 - 8	N1
N8 + 16	N32
N8 - 16	N56
N8 + 32	N42

B-Chord Routing Example - Ring



B-Chord Results

Number of entries in the finger table doubles

Average path length is expected to decrease to two-thirds that of traditional Chord

Analysis of Paper 3

Title

GA-Chord: An improvement to chord algorithm based on group autonomy in structured P2P Network

Authors

Chao Fan, Qing Liao, Jingling Zhao

Publication Year: 2010

Journal: 2010 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT)

Page Numbers: 1001 - 1004

URL: http://ieeexplore.ieee.org/search/srchabstract.jsp?tp=&arnumber=5705239

GA-Chord Motivation

- Chord is a sensitive system, the system spends much energy on re-establishing routing table and resource relocation.

- The GA-Chord algorithm tries to improve the lookup efficiency acheived.

GA-Chord Overview

- Improves query performance by making use of Group autonomy and Bi-directional lookup.
- A set of stable/powerful nodes are selected as leader nodes, that are responsible for groups of nodes
- Leaders form a logical outer ring
- Each node maintains a Neighbor table and a Finger table
- Queries are routed across groups using the Neighbor table, and within a group using the finger table

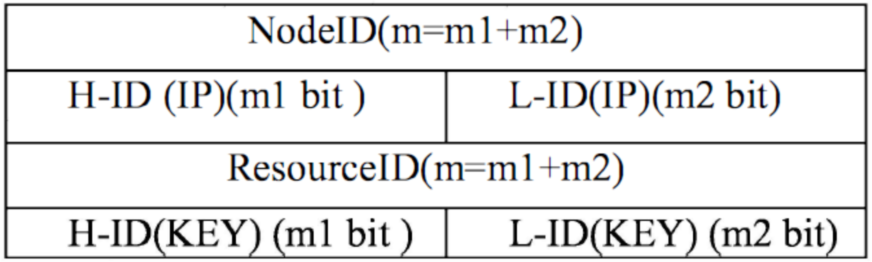
GA-Chord Overview cont...

It picks a few closed node group to be its group leaders.

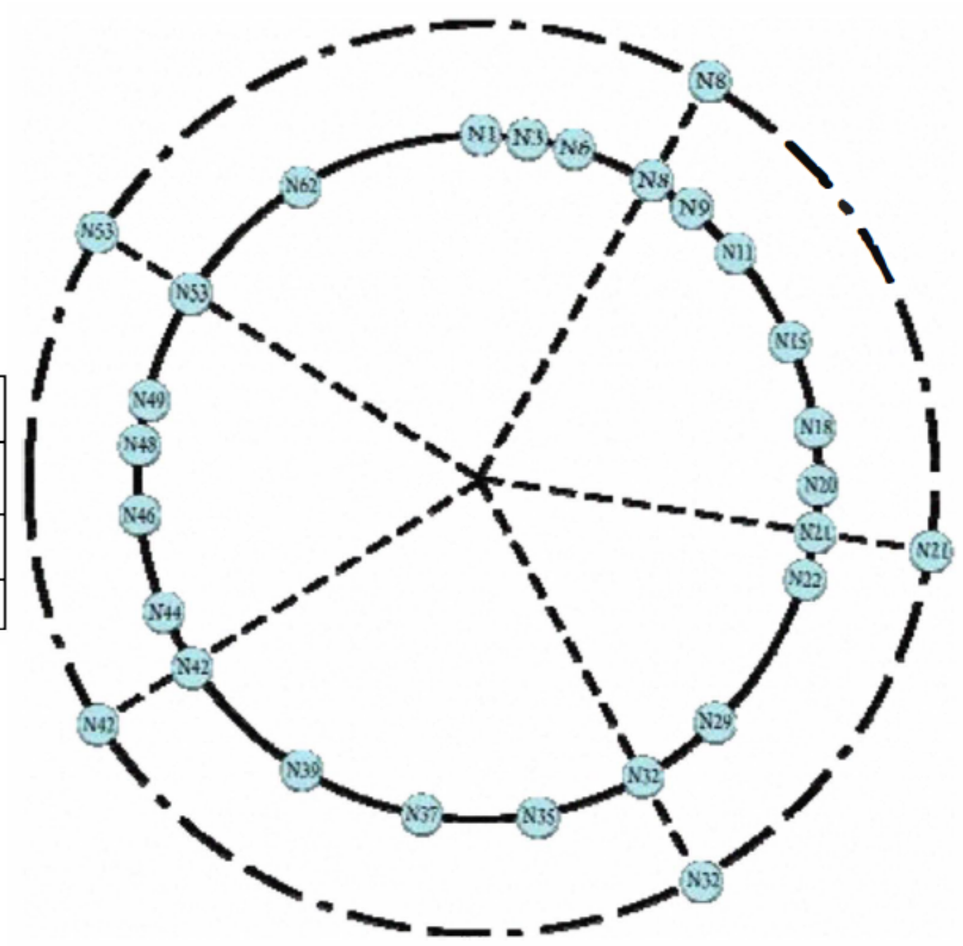
The nodes between these leaders fall under the same group.

Any query or modification in the system first goes to the particular group then it is tackled there.

GA-Chord Example - Ring



NodeID(m=m1+m2)	
H-ID (IP)(m1 bit)	L-ID(IP)(m2 bit)
ResourceID(m=m1+m2)	
H-ID(KEY) (m1 bit)	L-ID(KEY) (m2 bit)



Nodes entering and leaving

New node:

- When a new node enters it contacts its neighboring nodes and calculates its neighbor table and finger table.
- All nodes within the group update their neighbor table and finger table.
- The new node contends to act as the backup leader node.

Node leaves:

- When an inside node leaves, the nodes within the group update their neighbour tables and finger tables.
- If the leaving node is a leader node and a backup node exists, then it becomes the new leader else the next node anticlockwise becomes the leader

GA Chord Results

With a fixed group size, the average routing hop is greatly improved when compared to the traditional Chord.

With a fixed total number of nodes, increasing the group size from 1 (equivalent to Chord) to 10, reduces routing hops.

Software Design

1. Test case setup
2. Test case run
3. Request routing

Simulation Software Components

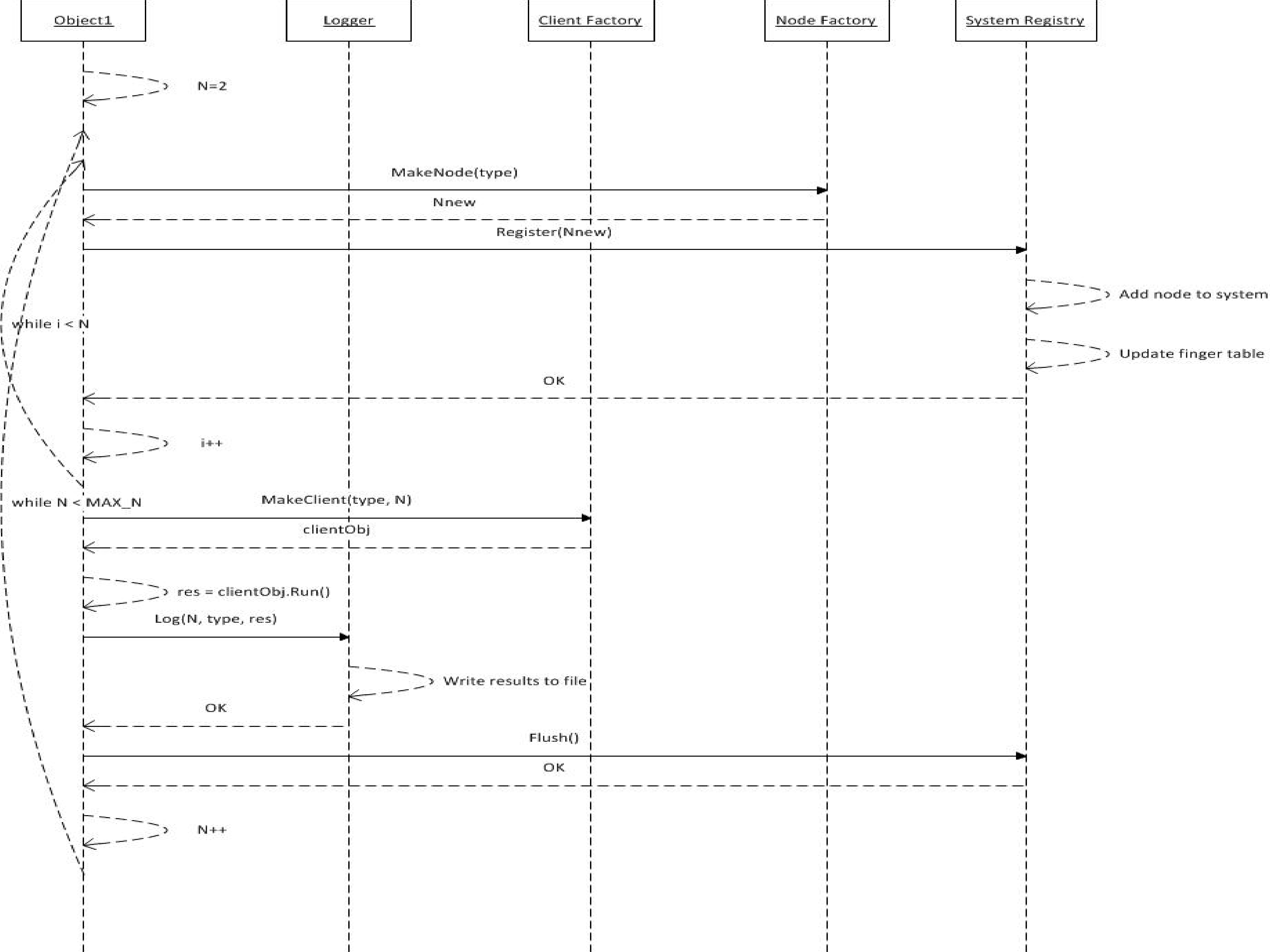
1. Nodes in the distributed system implementing the three protocols.
2. Software to setup the test environment, run tests and return results to log system.
3. Logging software to record the results to permanent storage.
4. Software to analyse the recorded results, plot graphs and verify hypothesis.

Test Case Setup

Sets up the infrastructure to run the test cases

Does the following:

- Creates a distributed system with N nodes using the selected lookup protocol
- Sets up a client to run the test case
- Sets up a logger to record the test results

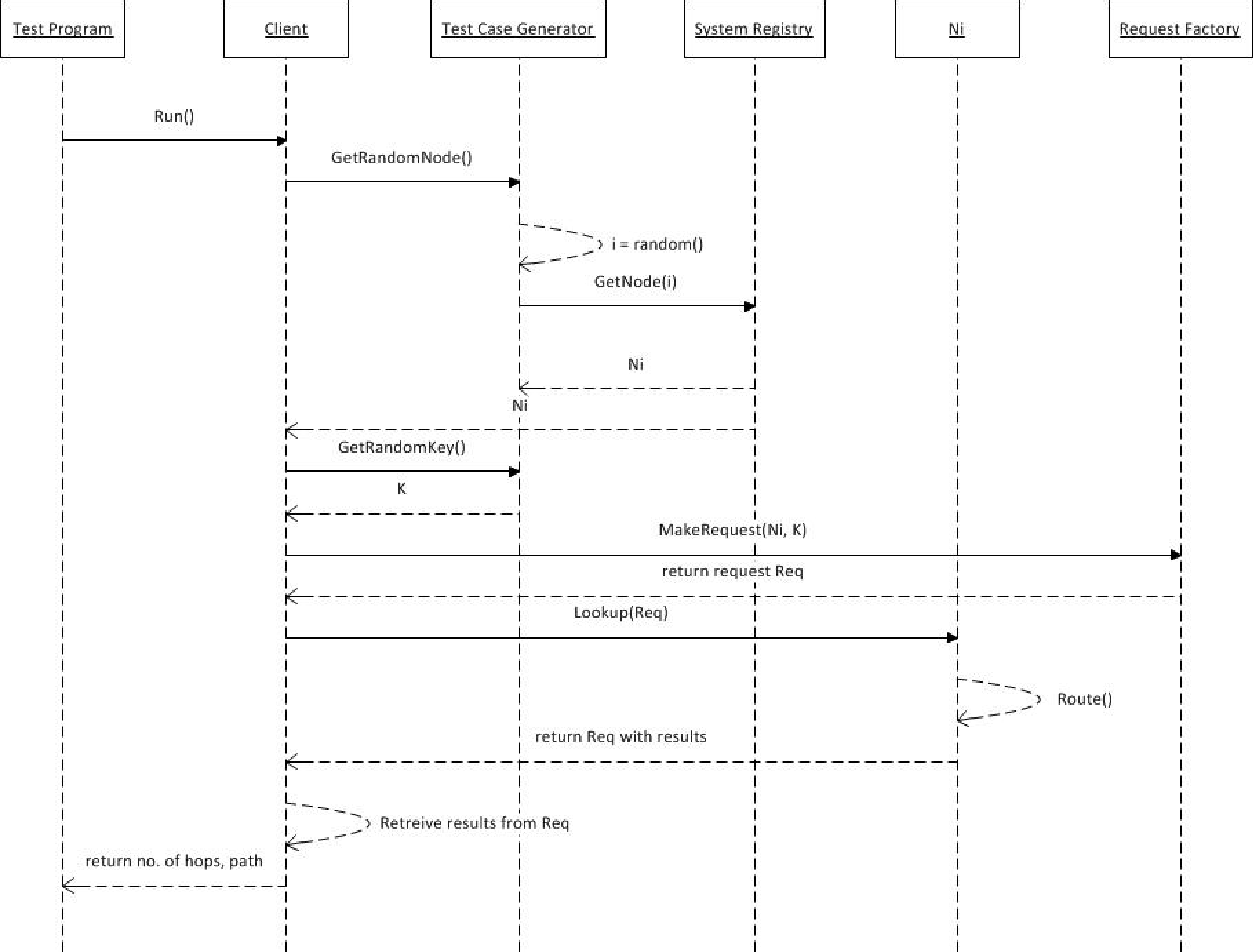

Object1
Logger
Client Factory
Node Factory
System Registry
N=2
MakeNode(type)
Nnew
Register(Nnew)
Add node to system
while i < N
Update finger table
OK
i++
while N < MAX_N
MakeClient(type, N)
clientObj
res = clientObj.Run()
Log(N, type, res)
Write results to file
OK
Flush()
OK
N++

Test Case Run

Runs the test case and records the the results.

Does the following:

- randomly select a initiating node
- randomly select a target key
- perform the lookup
- record the number of hops and the path taken


Test Program
Client
Test Case Generator
System Registry
Ni
Request Factory
Run()
GetRandomNode()
i = random()
GetNode(i)
Ni
Ni
GetRandomKey()
K
MakeRequest(Ni, K)
return request Req
Lookup(Req)
Route()
return Req with results
Retreive results from Req
return no. of hops, path

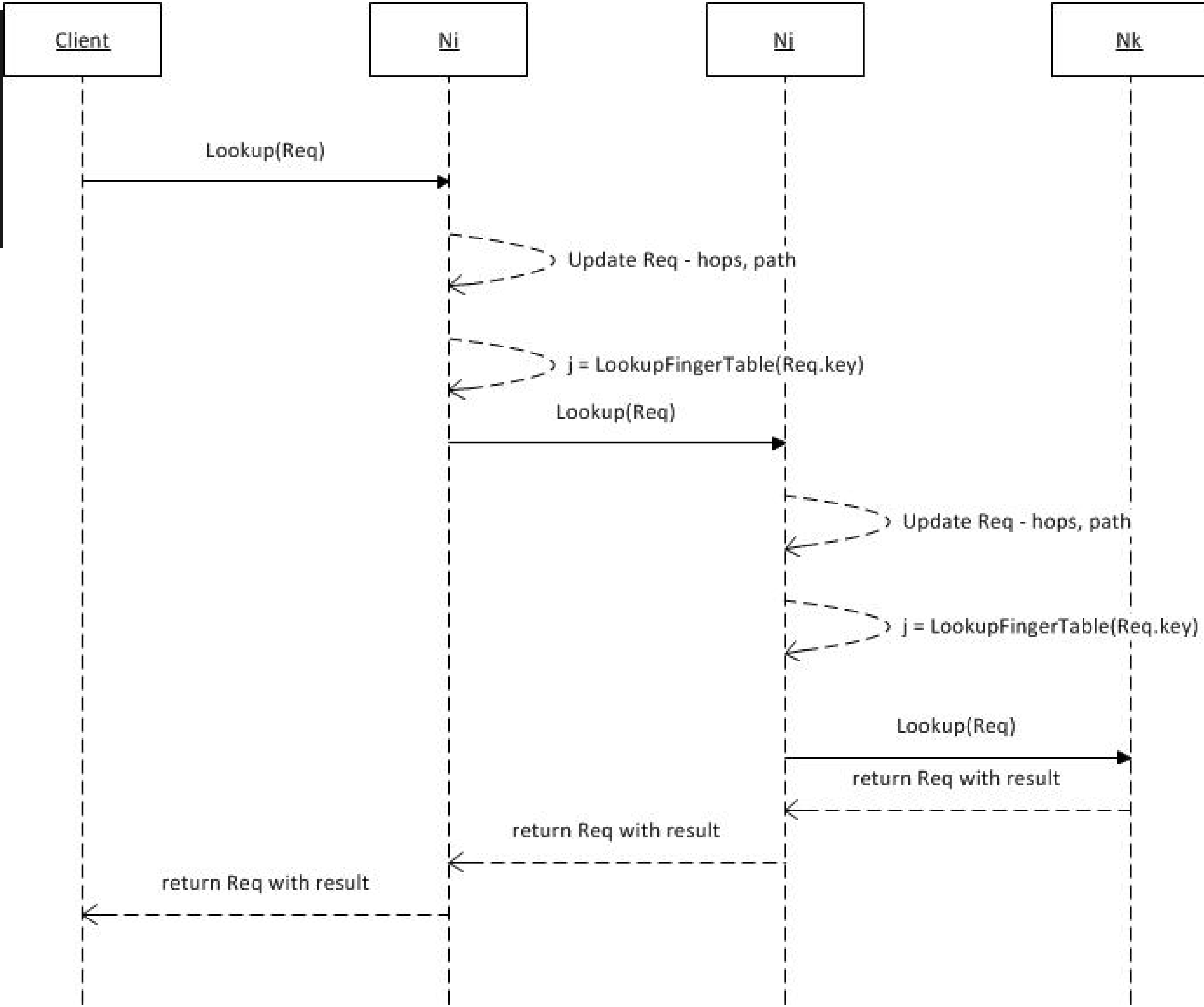
Request Routing

- Performs the routing of the request to the node that holds the target key.

- Returns the requested resource

- Keeps track of the number of hops and the path taken

- Implementation of routing and path taken depends on the protocol used by the node.


Client
Ni
Nj
Nk
Lookup(Req)
Update Req - hops, path
j = LookupFingerTable(Req.key)
Lookup(Req)
Update Req - hops, path
j = LookupFingerTable(Req.key)
Lookup(Req)
return Req with result
return Req with result
return Req with result

Analyse logged data

- Results of each test case run is recorded in separate files for each protocol
- Software is created to analyse the recorded result data
- Graphs are drawn to study the relationship between the number of nodes and the average number of hops
- Analysis of data is expected to verify the hypothesis

Next Steps

- Class diagram for implementation

- Implementation of Chord, B-Chord and GA-Chord nodes

- Framework to run tests and record results using the Chord and the B-chord protocols

Thank you

Questions ?