Computer Networks (CS F303)

BITS Pilani
Pilani Campus

Virendra Singh Shekhawat
Department of Computer Science and Information Systems

BITS Pilani
Pilani Campus

Second Semester 2020-2021
Module-2 Application Layer

Today's Agenda

- P2P Applications
- Distributed Hash Tables
- Circular DHT Protocol
  – Chord Protocol

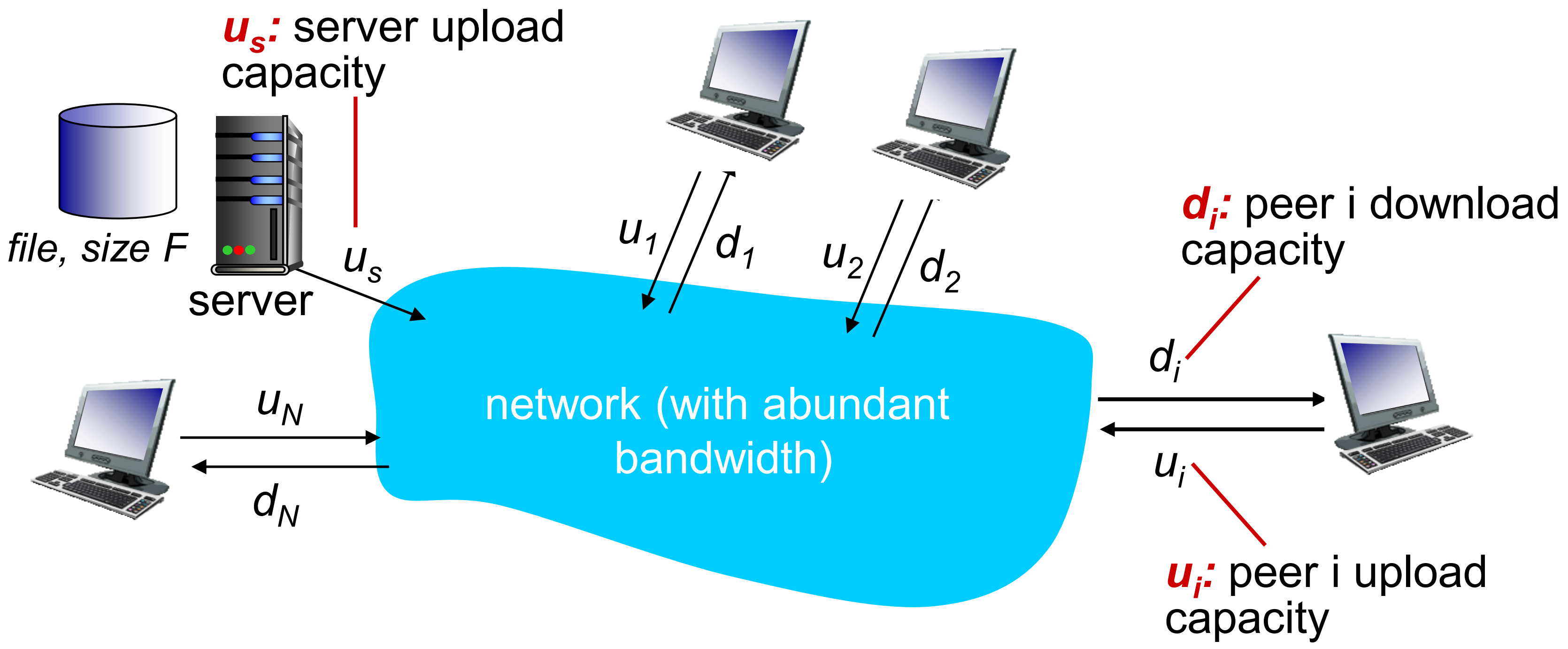
Peer to Peer (P2P) Architecture

- *No* always-on server
- Arbitrary end systems directly communicate
- Peers are intermittently connected
- Examples
  – File distribution (BitTorrent)
  – Streaming (KanKan)
  – VoIP (Skype)

File Distribution: P2P vs CS

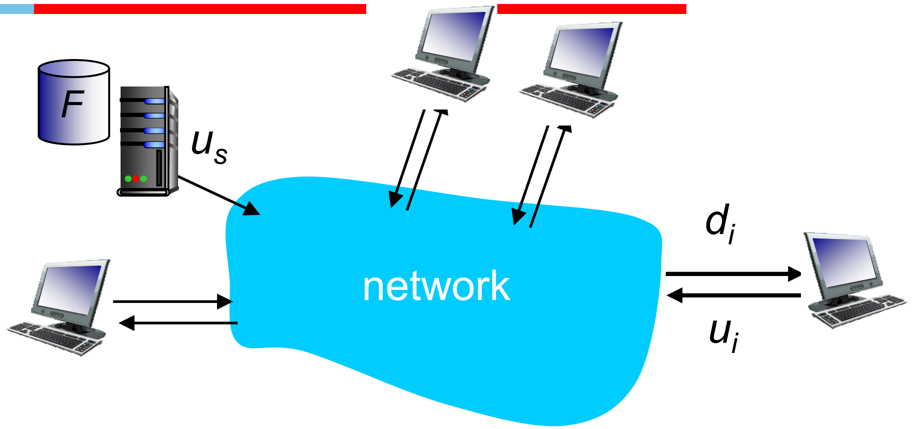
Q: How much time to distribute file (size *F*) from one server to *N peers*?

– peer upload/download capacity is limited resource



File Distribution Time

- *Server transmission:* must sequentially send (upload) *N* file copies:
  - Time to send N copies: NF/u_s

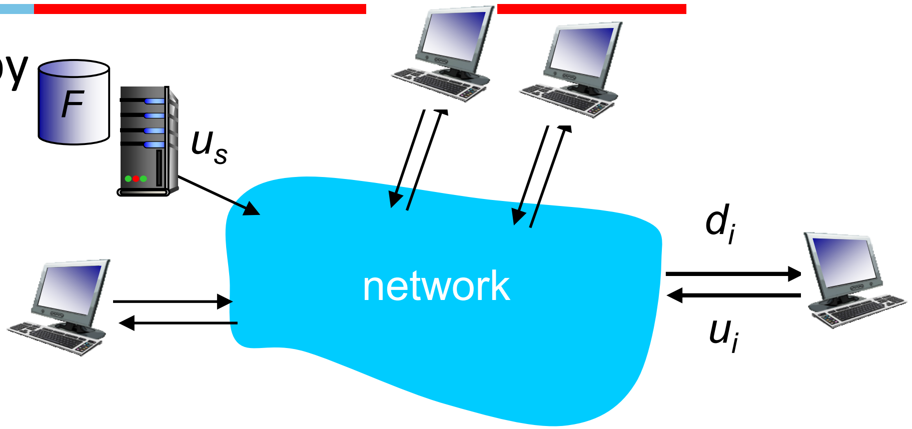


❖ *Client:* each client must download file copy
  - d_{min} = min client download rate
  - Slowest client download time: F/d_{min}

time to distribute F to N clients using client-server approach	$D_{c\text{-}s} \geq max\{NF/u_{s,},F/d_{min}\}$

File Distribution Time: P2P

- *Server transmission:* must upload at least one copy
  - time to send one copy: F/u_s



- *Client:* each client must download file copy
  - Slowest client download time: F/d_{min}
- *Clients:* as aggregate must download *NF* bits
  - max upload rate (limiting max download rate) is $u_s + \Sigma u_i$

Time to distribute F to N clients using P2P approach

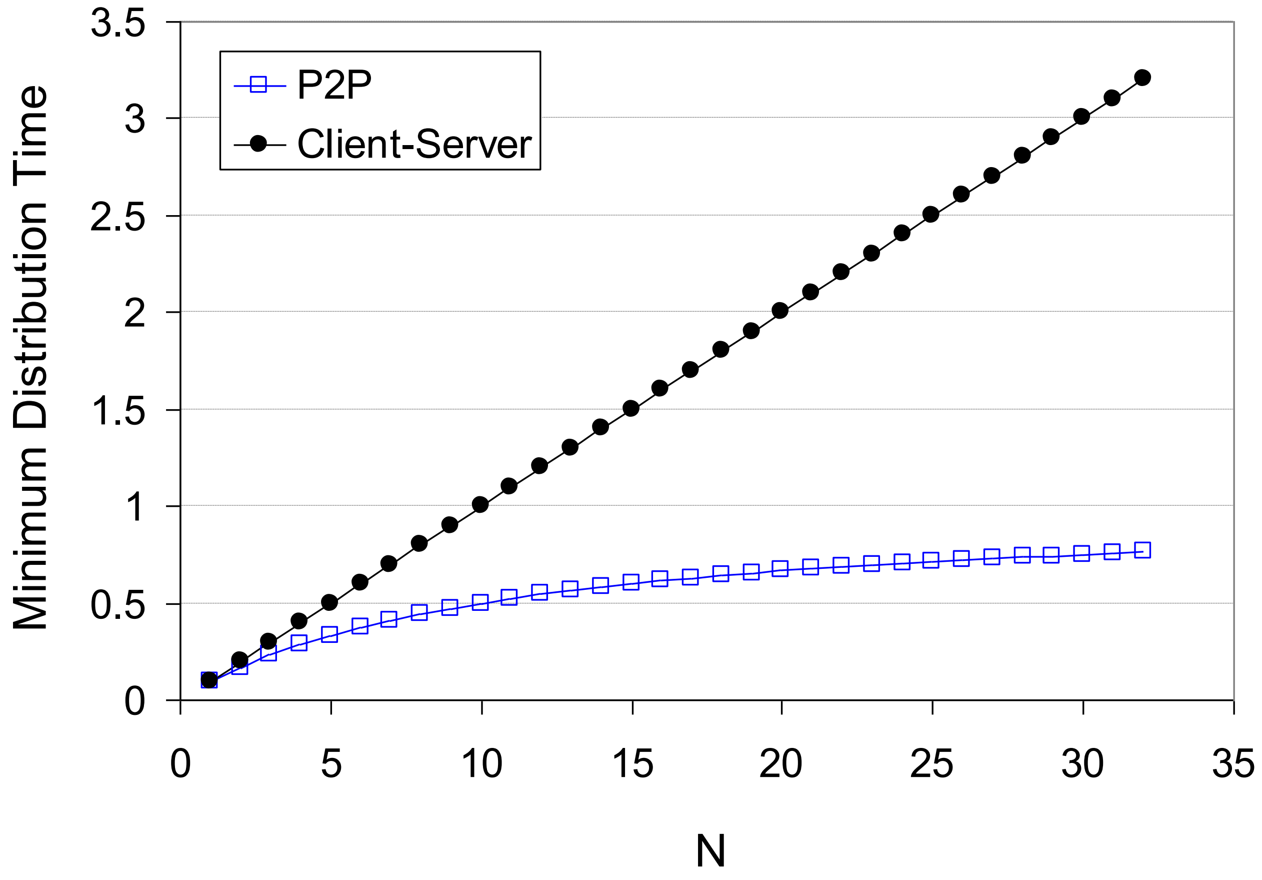
$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

Exercise

- Distributing a File F = 15 Gbits to 10 peers
- Server upload rate is u_s = 30 Mbps
- Each peer download rate is d_i = 2 Mbps
- Each peer upload rate is u = 300 Kbps
- Question
  – Calculate minimum distribution time for both CS and P2P

CS vs P2P: Example

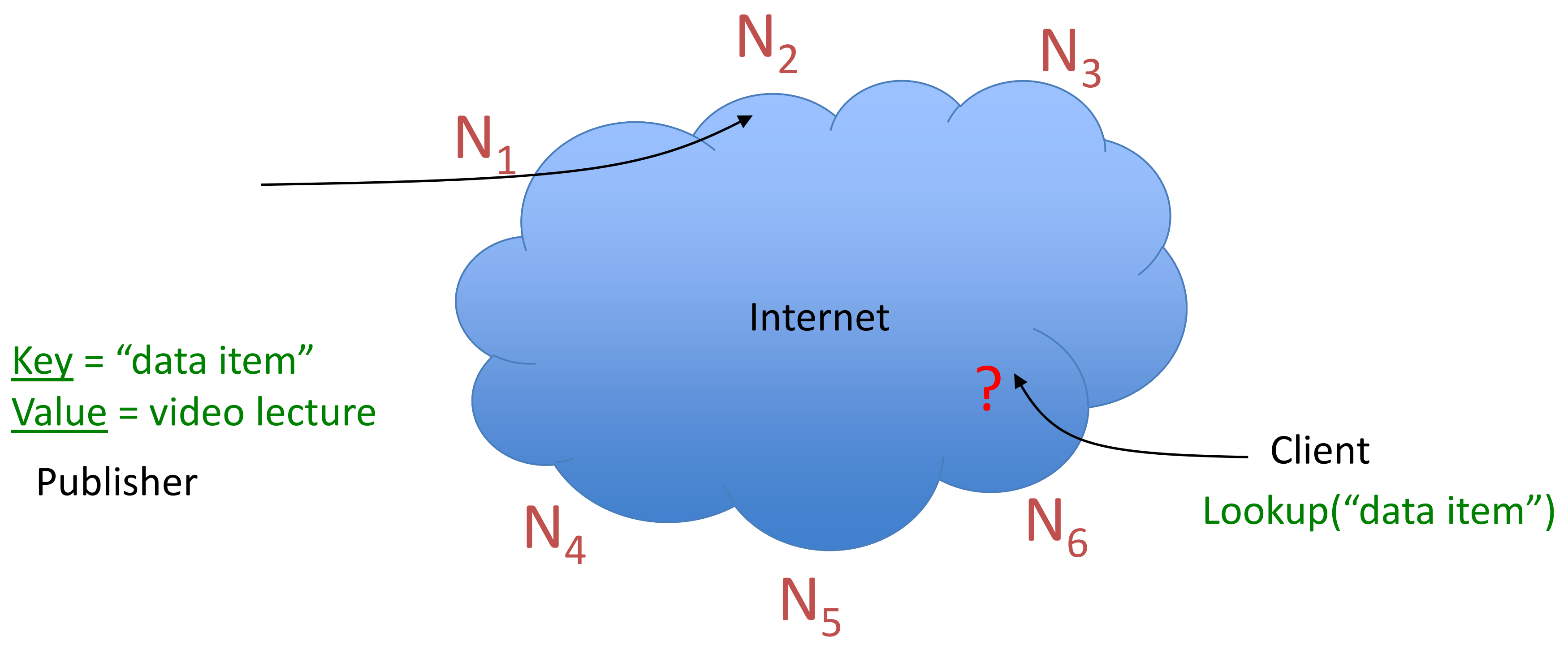
client upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$



P2P File Distribution: BitTorrent

- File divided into 64 KB to 1 MB size (typically 256 KB) chunks
- Trackers
  - Tracks peers participating in torrent
  - New peer joins torrent and registers with tracker to get list of peers, connects to subset of peers
- Torrent
  - Group of peers exchange chunks of a particular file
- Peers in torrent send/receive file chunks
  - At any given time, each peer will have a subset of chunks from the file
  - A peer asks its neighbors for the list of chunks they have and gets list from each
  - A peer needs to take a call on-
    - Which chunks should it request first from its neighbor?
    - To which of its neighbors it should send requested chunks?

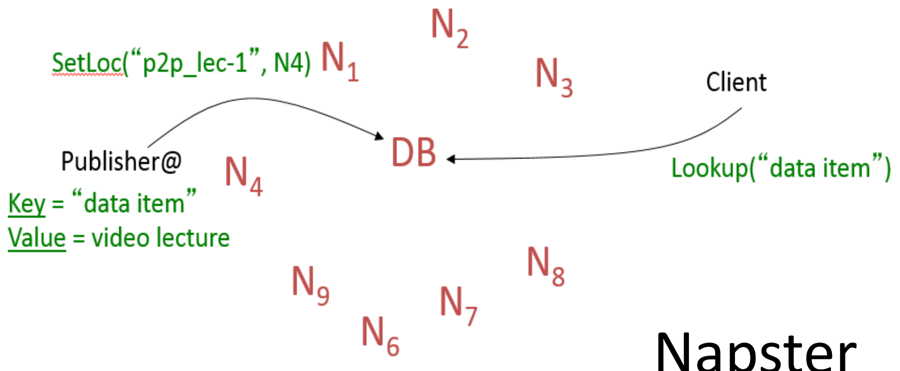
The lookup problem



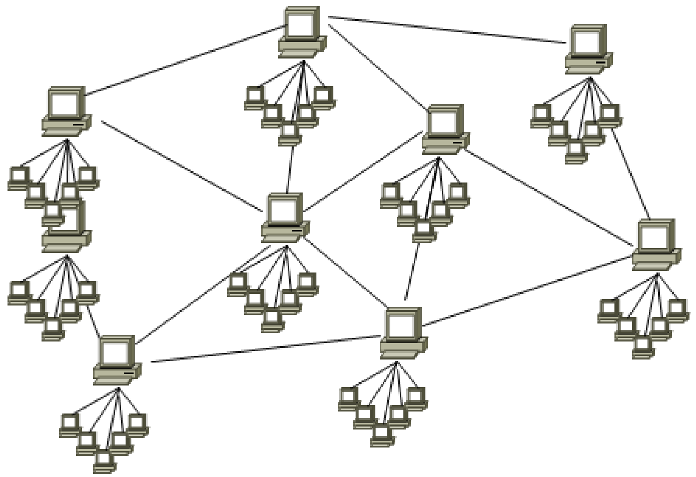
Decentralized network with several peers (servers/clients)

How to find specific peer that hosts desired data within this network?

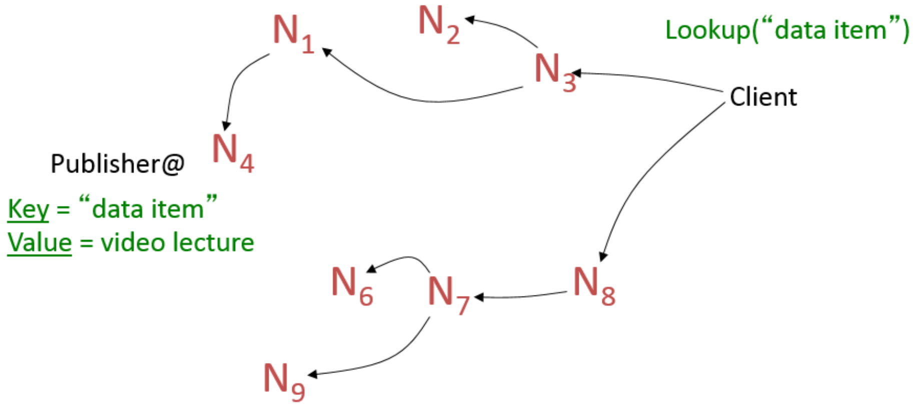
P2P Protocols



Napster



Kazaa (Skype is based on Kazaa)



Gnutella

Distributed Database

- Each Peer hold a small subset of the total (key, value) pairs

- Any Peer can query the distributed database with a particular key
  - Distributed DB locate the Peers that have the corresponding (key, value) pairs and return to the querying Peer
  - Any Peer can insert new (key, value) pairs into the DB

Distributed Hash Table Implementation [.1]

- Randomly scatter the (key, value) pairs across all the peers
- Each peer maintain a list of the IP addresses of all peers
- The querying peer sends its query to all other peers
- The peers containing the (key, value) pairs that match the key can respond with matching pairs
- This approach is not scalable. Why?

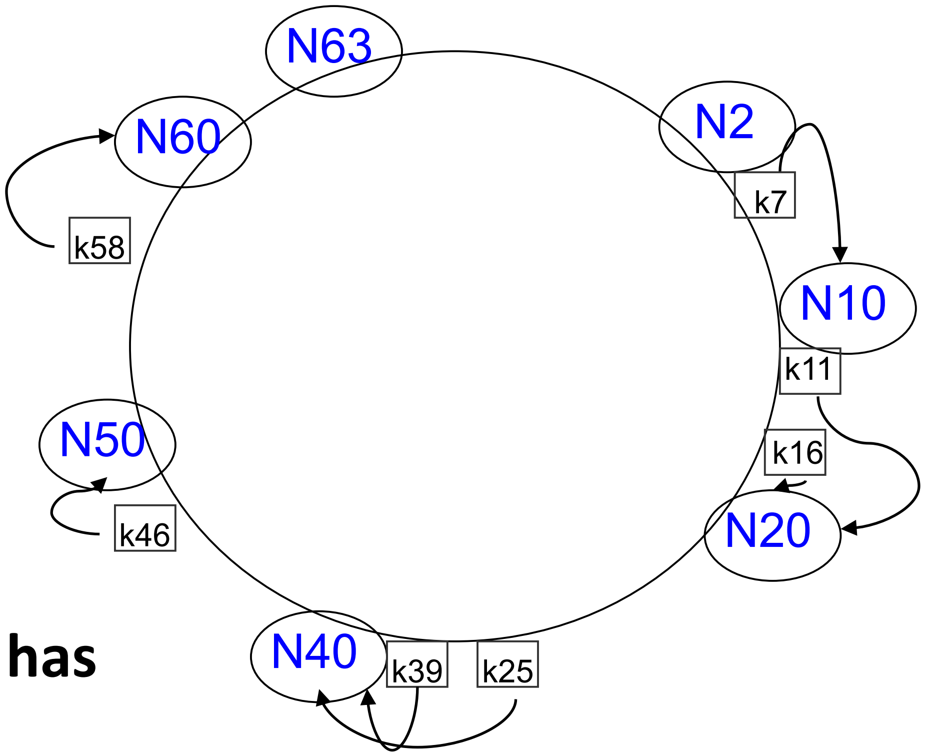
Database Implementation [..2] Circular DHT

- Hash function assigns each “**node**” *and* “**key**” an m-bit *identifier* using a base hash function such as SHA-1
  - Node_ID = hash(IP, Port)
  - Key_ID = hash(original key)

ID Space: 0 to 2^m-1

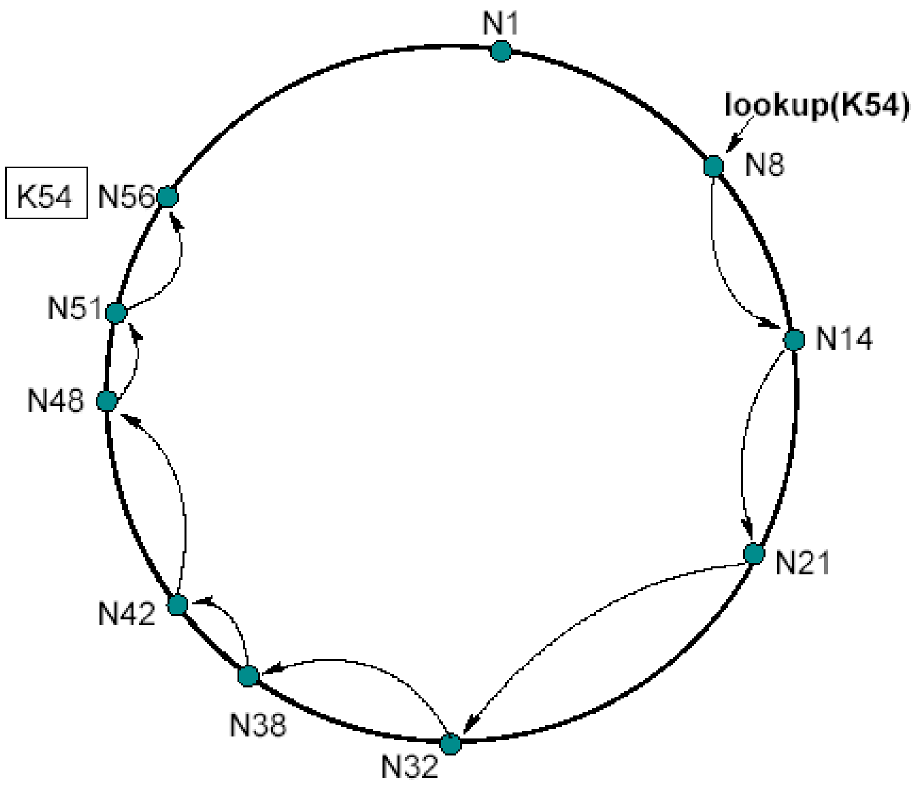
Here: *m = 6*

Range = 64



Assign (key-value) pair to the peer that has the *closest* ID.

Chord Protocol:Lookup Operation Example



Predecessor: pointer to the previous node on the id circle

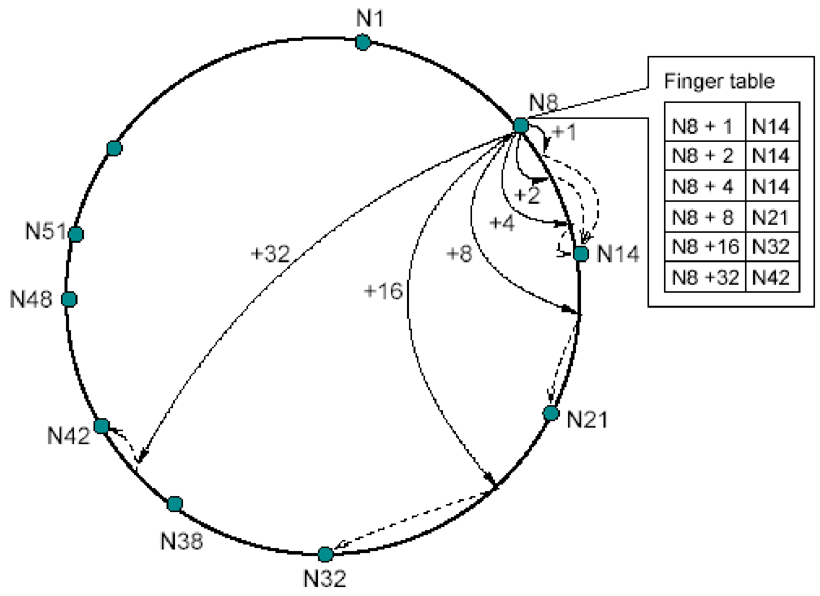
Successor: pointer to the succeeding node on the id circle

- ask node ***n*** to find the successor of ***id***
- If ***id*** between ***n*** and its ***successor***
  return ***successor***
- else forward query to ***n´***s ***successor*** and so on

=>#messages linear in #nodes

Scalable node localization

- Each node n contains a routing table with up-to m entries (**m**: number of bits of the identifier) => **finger table**
- i^{th} entry in the table at node **n** contains the first node **s** that succeds **n** by at least $\mathbf{2^{i-1}}$
  - **s = successor (n + $\mathbf{2^{i-1}}$)**
  - **s** is called the ith finger of node **n**

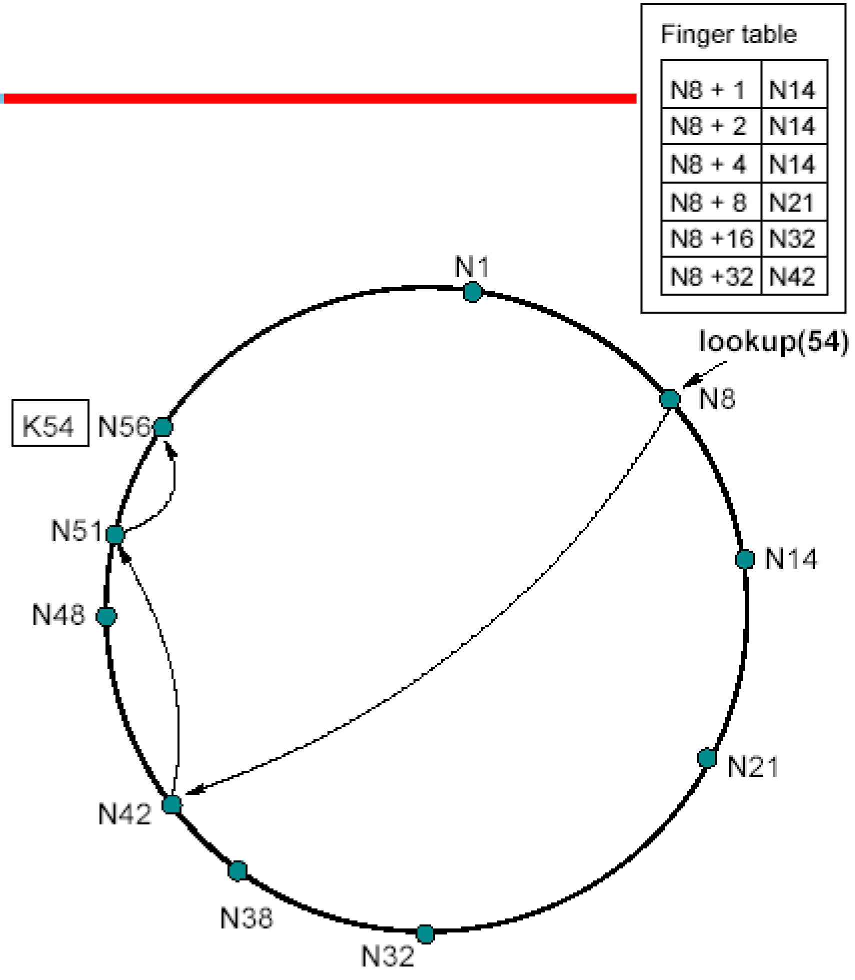


The Chord algorithm – Scalable node localization

- Search in finger table for the node which is

  most immediatly precedes key

- Invoke **find_successor** from that node



Number of messages O(log N)!

Failure Recovery (Peer Churn)

- Key step in failure recovery is maintaining correct successor pointers
- To achieve this, *each node maintains a successor-list* of its *r* nearest successors on the ring
- *If node n notices that its successor has failed,* it replaces it with the first live entry in the list
- The ***stabilize*** will correct finger table entries and successor-list entries pointing to failed node
- Stabilization protocol should be invoked based on the frequency of nodes leaving and joining

Thank You!