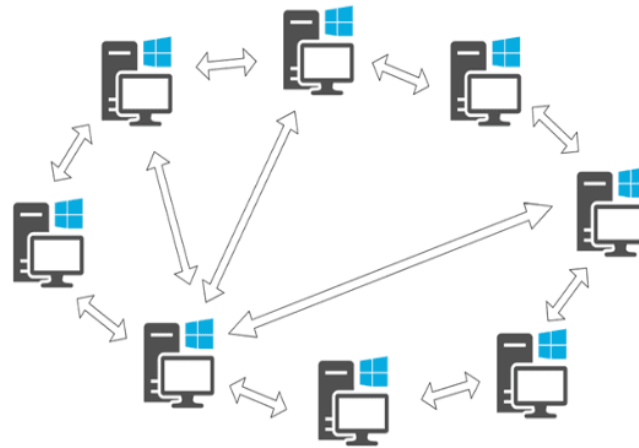


P2P Applications



Dr. Manas Khatua

Assistant Professor

Dept. of CSE, IIT Guwahati

E-mail: manaskhatua@iitg.ac.in

Peer-to-Peer Applications

- Web, E-mail, FTP, DNS – all employ **client-server** architectures
- With a P2P architecture
 - there is **minimal** (or no) **reliance on always-on infrastructure** i.e. servers.
 - pairs of intermittently connected **hosts communicate directly** with each other.
- Few P2P applications
 - **File Distribution**: the application distributes a file from a single source to a large number of destinations
 - BitTorrent system
 - **Distributed database**: a database distributed over a large community of peers
 - Distributed Hash Table (**DHT**).

File Distribution

- Let the **task** is of
 - distributing a large file
 - from a single server to a large number of hosts
- The **file** might be
 - a new version of OS / a software patch / an MP3 music file / an MPEG video file

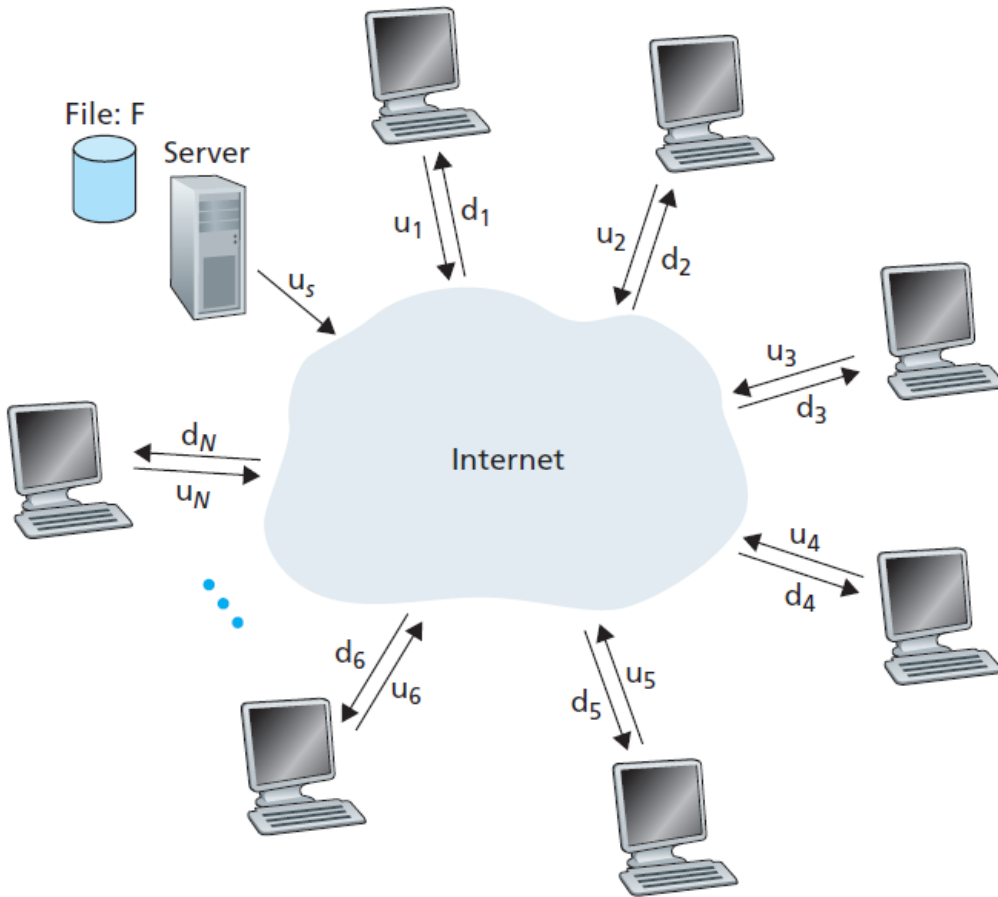
client-server file distribution: server must send a copy of the file to each of the peers.

- places enormous burden on the server
- consuming a large amount of server bandwidth

P2P file distribution: each peer can redistribute any portion of the file it has received to any other peers.

- assisting the server in the distribution process

Scalability of P2P Architectures



- **upload rate** of the i -th peer's access link is u_i
- **upload rate** of the server's access link is u_s
- **download rate** of the i -th peer's access link by d_i .
- the **size of the file** to be distributed (in bits) is F
- the **number of peers** that want to obtain a copy of the file is N .
- The **distribution time** is the time it takes to get a copy of the file to all N peers
- Let the Internet core has enough bandwidth

Cont... (distribution time)

In client-server architecture:

- **Step1**: server must transmit one copy of the file to each of the N peers. the time to distribute the file must be at least NF/u_s .
- **Step2**: Let $d_{\min} = \min\{d_1, d_p, \dots, d_N\}$.
- The peer with the lowest download rate cannot obtain all F bits of the file in less than F/d_{\min} seconds.
- So, the **distribution time** for the *client-server architecture* is

$$D_{cs} \geq \max \left\{ \frac{NF}{u_s}, \frac{F}{d_{\min}} \right\}$$

- For N large enough, the client-server distribution time is given by NF/u_s
- Thus, distribution time **increases linearly with** the number of peers N

Cont... (distribution time)



In P2P architecture:

- In P2P, when a peer receives some file data, it can use its own upload capacity to redistribute the data to other peers.

Calculating the distribution time for the P2P architecture is somewhat more complicated.

A simple expression for the minimal distribution time:

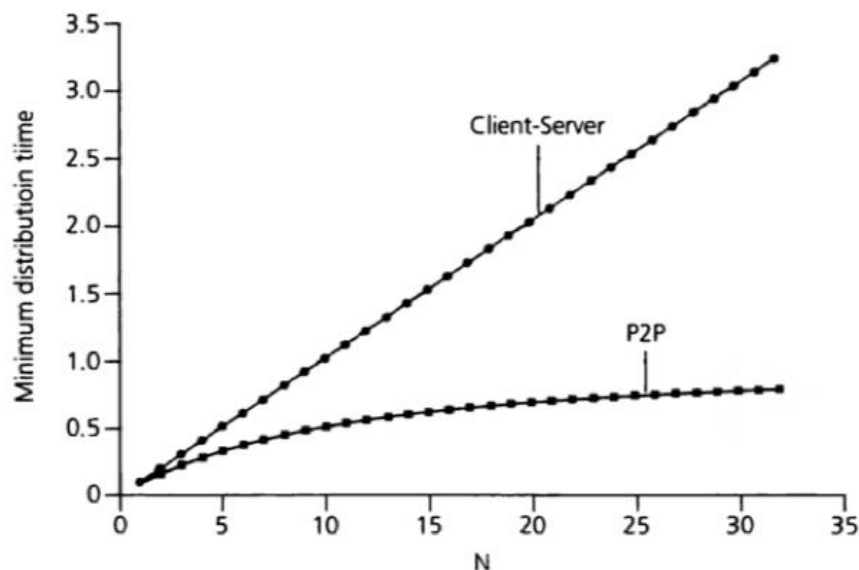
- **Step1:** At the beginning, only the server has the file.
- To get this file *into the community* of peers, minimum distribution time is at least F/u_s .
- **Step2:** the peer with the lowest download rate cannot obtain all F bits of the file in less than F/d_{\min} seconds.
- **Step3:** the total upload capacity of the system is, $u_{\text{total}} = u_s + u_1 + \dots + u_N$.
 - The system must deliver (upload) F bits to each of the N peers.
 - This cannot be done at a rate faster than u_{total} .
 - So, the minimum distribution time is also at least $NF/(u_s + u_1 + \dots + u_N)$.

- Finally, the **minimum distribution time for P2P**
- $$D_{\text{P2P}} \geq \max \left\{ \frac{F}{u_s}, \frac{F}{d_{\min}}, \frac{NF}{u_s + \sum_{i=1}^N u_i} \right\}$$

Cont...

- if we imagine that each peer can **redistribute** a bit **as soon as it receives** the bit, then there is a **redistribution scheme** that actually achieves this **lower bound**
- In reality, where chunks of the file are redistributed rather than individual bits, the above estimation serves as good approximation of distribution time.

Comparison of the minimum distribution time for the client-server and P2P



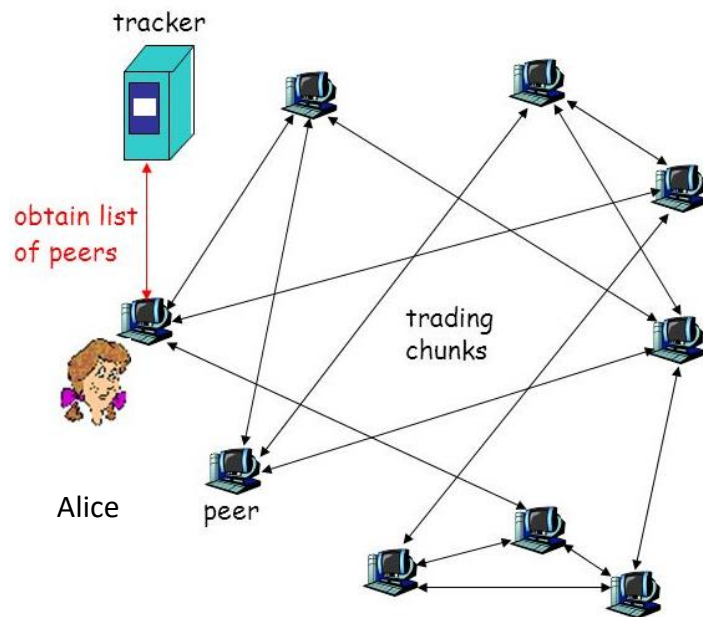
BitTorrent



- BitTorrent is a popular **P2P protocol** for file distribution
- the **collection of all peers** participating in the distribution of a particular file is called a **torrent**.
- Peers in a torrent download **equal-size chunks** of the file from one another (typical chunk size = 256 KBytes).
- When a peer first joins a torrent, it **has no chunks**.
- Over time it **accumulates** more and more **chunks**.
- Any peer may **join/leave** the torrent at any time.
- P2P **live streaming applications**, such as PPLive and ppstream, have been **inspired by** BitTorrent

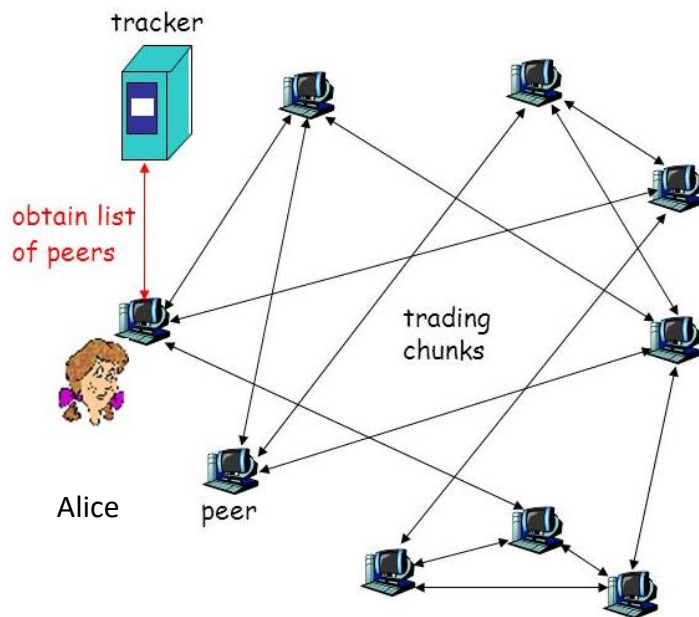
Cont...

- Each torrent has an **infrastructure node** called a **tracker**
 - Each peer registers itself with the tracker
 - Each peer periodically informs the tracker that it is alive
- So, tracker keeps track of all the registered peers



- After registering with torrent,
 - Alice gets a list of peers (subset of full torrent) from the tracker
 - Alice try to establish concurrent TCP connections with all peers of the list
 - Connected peers are called neighboring peers (see the figure – Alice has 3 neighbours at present)
 - A peer's neighboring peers will fluctuate over time.
 - Alice will ask each of her neighboring peers for the list of chunks they have
 - Alice will obtain L lists of chunks from L neighbors

Chocked/Unchocked Peers



- Alice will **issue requests** for a **chunk** she doesn't have.
- Alice uses **rarest first** technique for deciding the chunk she **needs first**
- Similarly, Alice also gets requests from other peers
- To determine which requests she responds to
 - a **clever trading algorithm** is used
 - gives **priority** to the neighbors that are **currently supplying her data** at the highest rate.
- Alice continually measures the rate and **determines the four peers** feeding her at highest rate.
 - these four peers are said to be **unchoked**

Randomly choose a **new trading partner**:

- Periodically she also **picks one additional neighbour at random** and sends it chunks.
- This peer is said to be **optimistically unchoked**
- **Objectives:** over the time this peer could include Alice in its unchoked list and then will send file with high rate to Alice
- **All other** neighboring peers **besides** these **five peers** (four "top" peers and one probing peer) are **"choked,"** that is, they do not receive any chunks from Alice.

Distributed Hash Tables (DHTs)

Goal: implement a distributed database in a P2P network

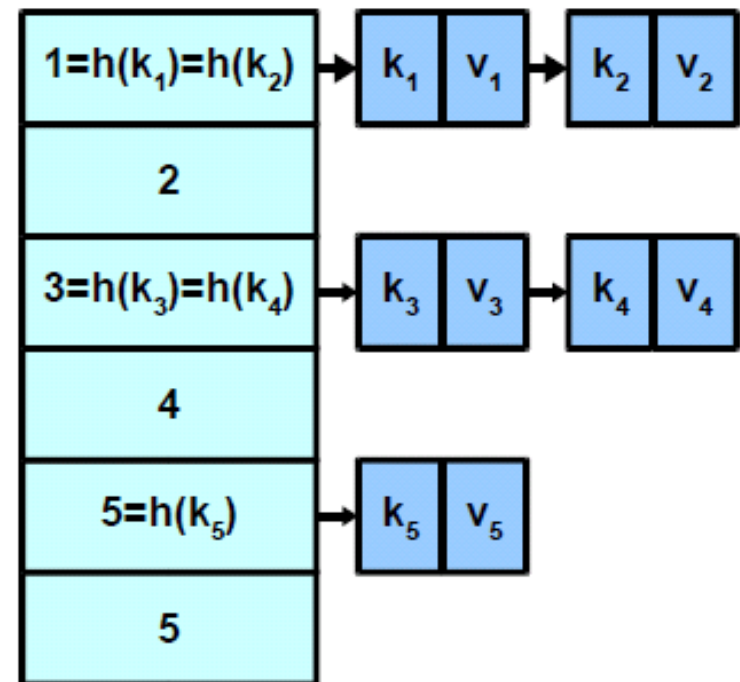
- **Hash Table:** data structure that maps “keys” to “values”
- Single-node hash table:
 - $\text{key} = \text{Hash}(\text{name})$
 - $\text{put}(\text{key}, \text{value})$
 - $\text{get}(\text{key}) \rightarrow \text{value}$
- How do I do this across millions of hosts on the Internet?

Ans: Distributed Hash Table

Example of A Hash Table

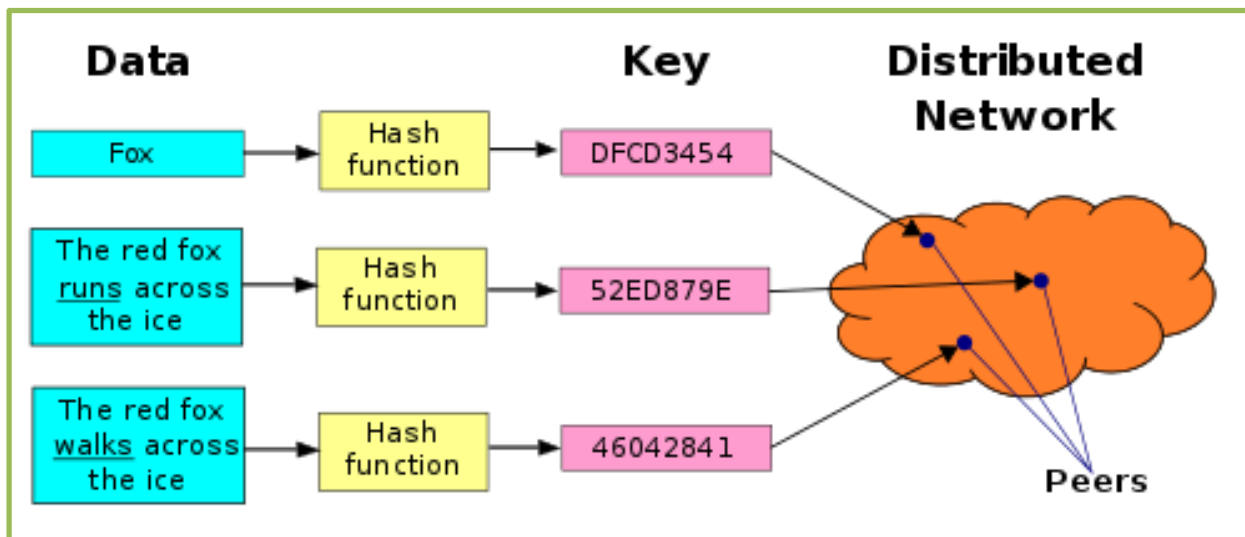
Allocated array:
indexed by hash
values

Stored entries



Cont...

- **DHT** is a *distributed system* that provides a **lookup service** similar to a hash table
 - (key, value) pairs are stored in a DHT, and
 - any participating node can efficiently retrieve the value associated with a given key.
- **Main advantage** of a DHT:
 - Nodes can be added/removed with minimum work
 - Responsibility for maintaining (key,value) mapping is distributed among the nodes



Distributed Hash Tables

Few distributed networks that use DHTs

- BitTorrent's distributed tracker
- Coral CDN
- Storm Botnet

DHT Interface

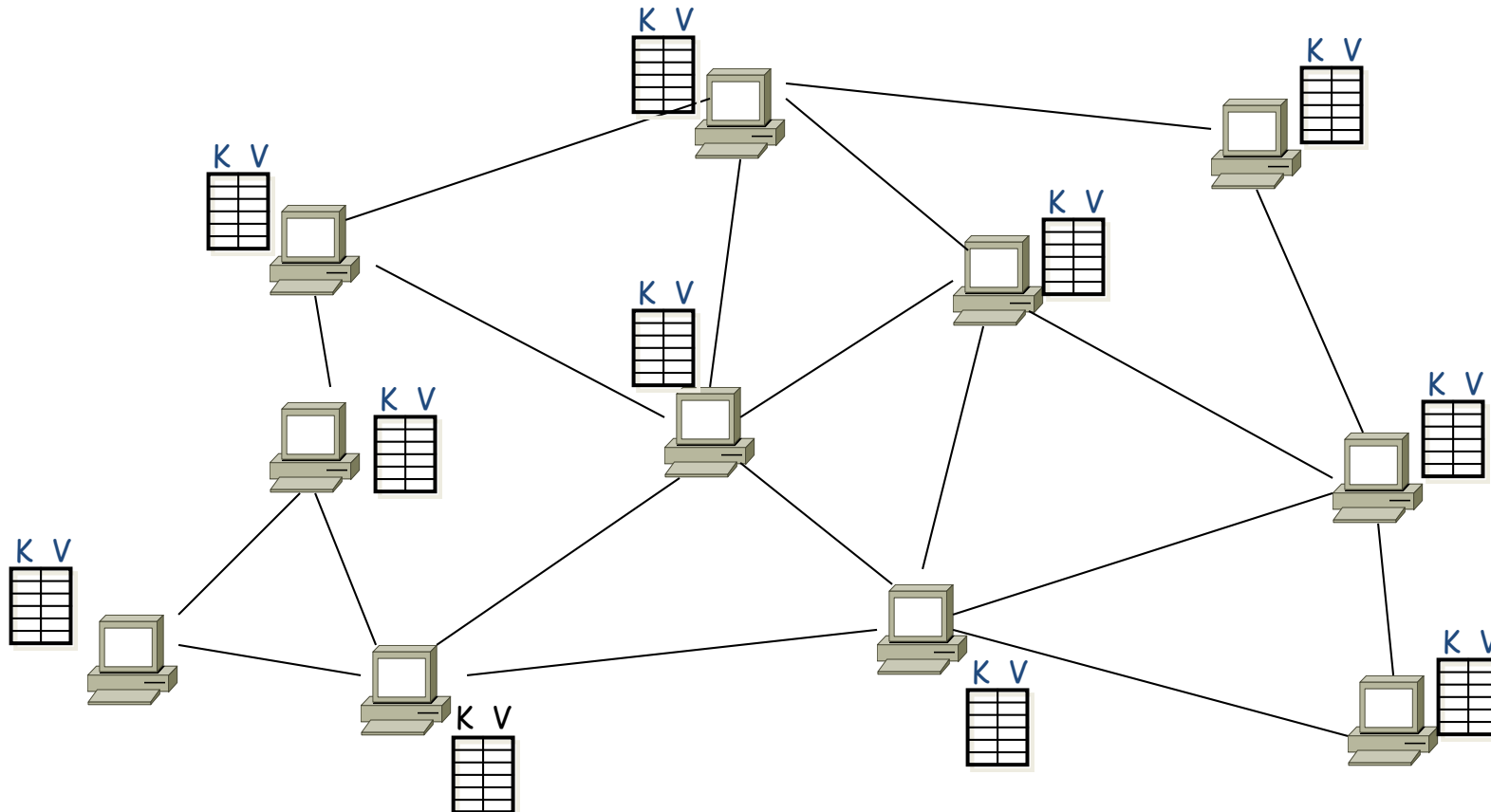
- insert(key, value)
- lookup(key)

Cont...



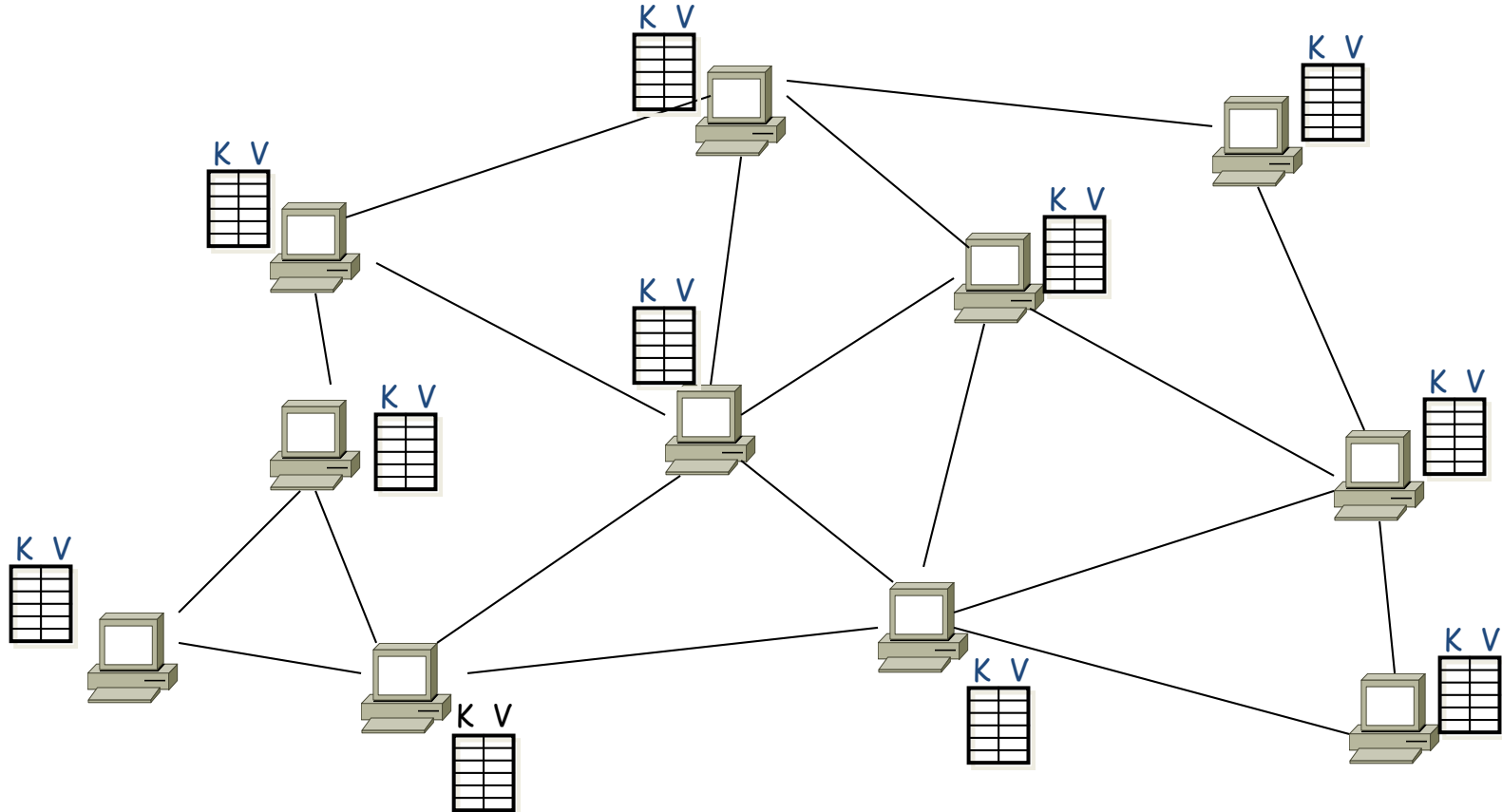
- In the P2P system, **each peer/node** will
 - only **hold** a small **subset of the totality** of the (key, value) pairs.
 - **route messages** to node holding the **key**
- Any **peer** is allowed to
 - **insert** new key-value pairs into the database
 - **query** the distributed database with a given **key**.
- The distributed database **application** will
 - **locate the peers** that have the corresponding (key, value) pairs,
 - **return the key-value pairs** to the querying peer.

How do DHTs work?



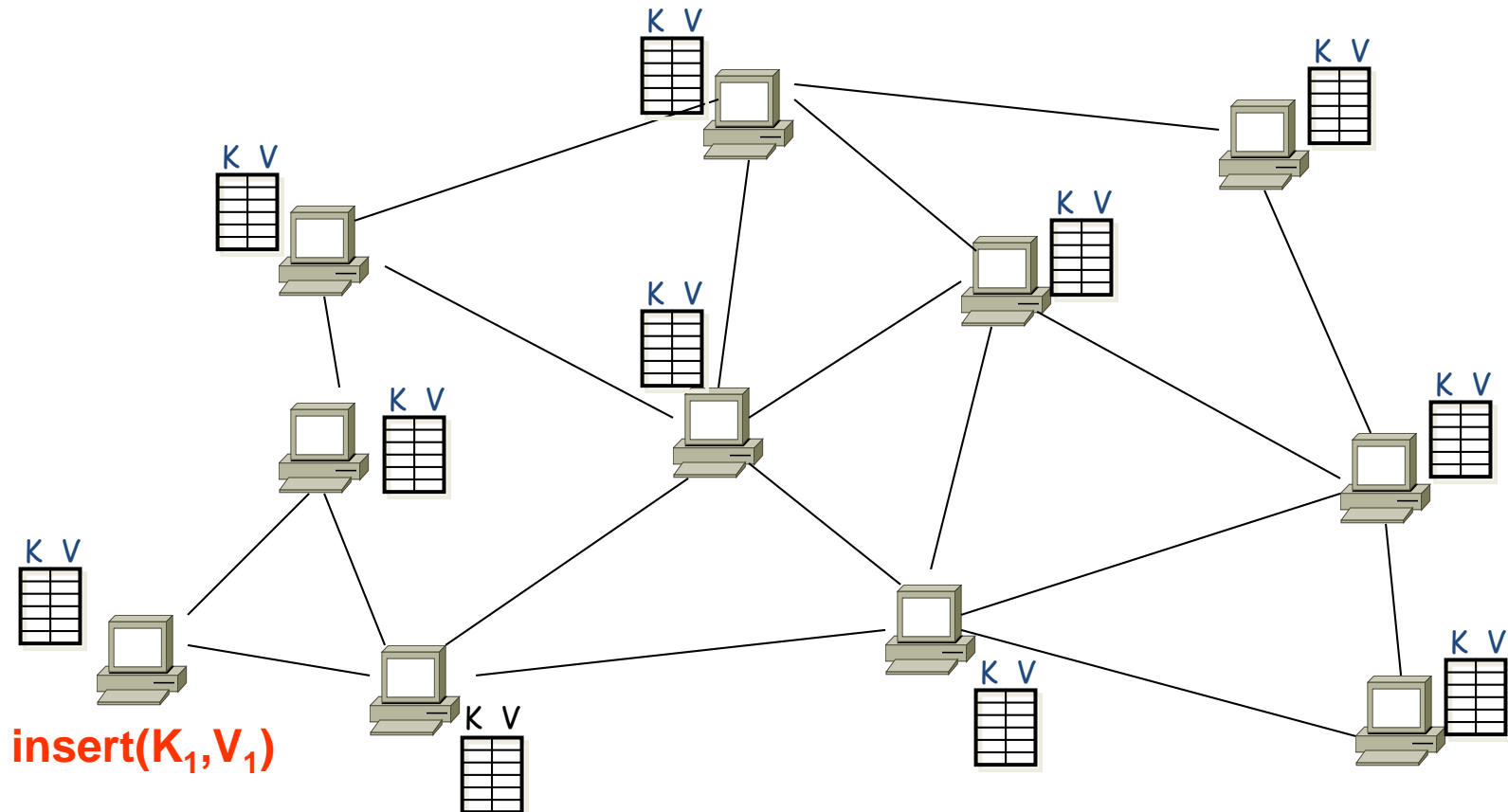
Neighboring nodes are "connected" at the application-level

Cont...



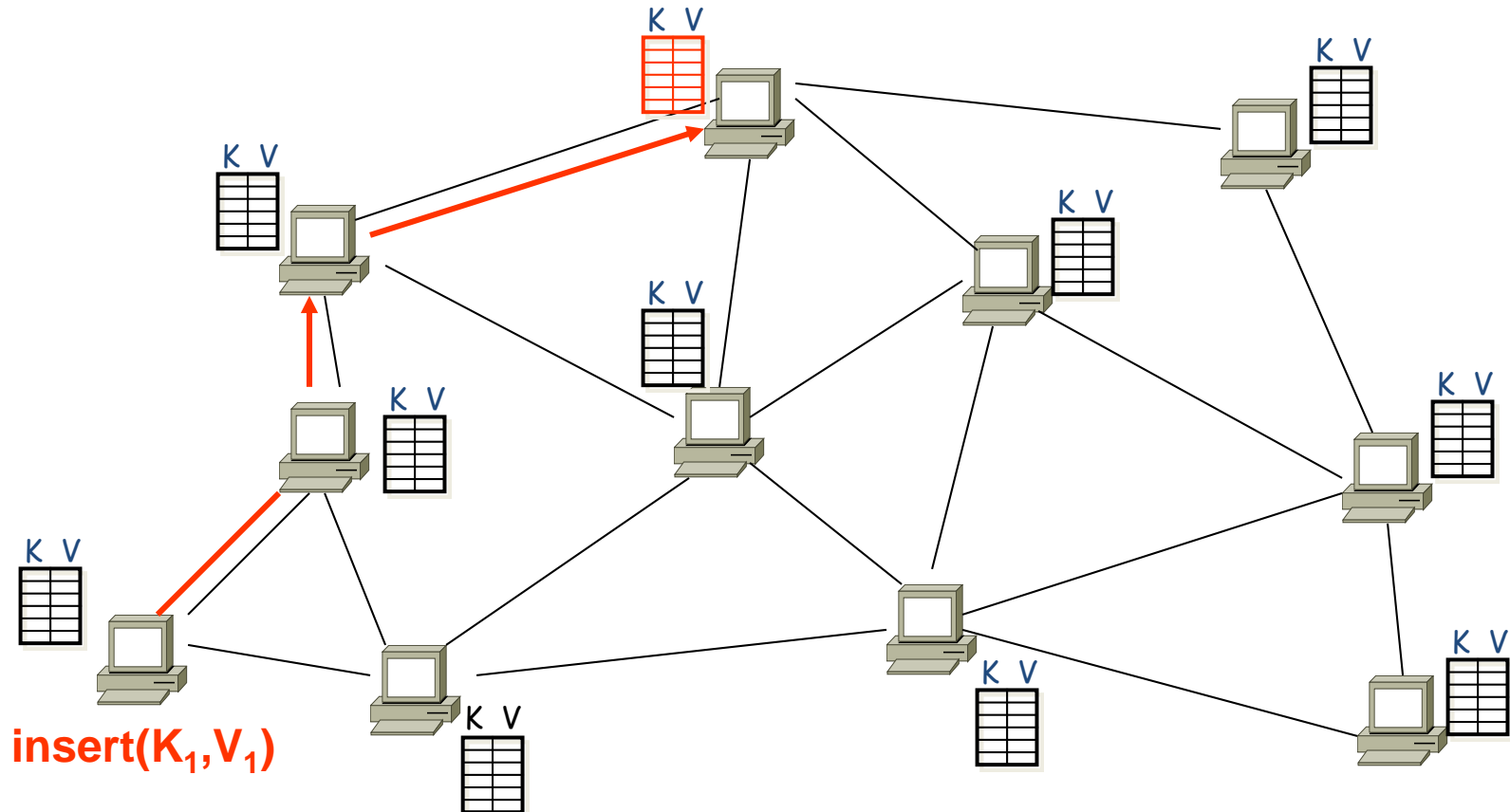
Operation: take *key* as input; route messages to node holding *key*

Cont...



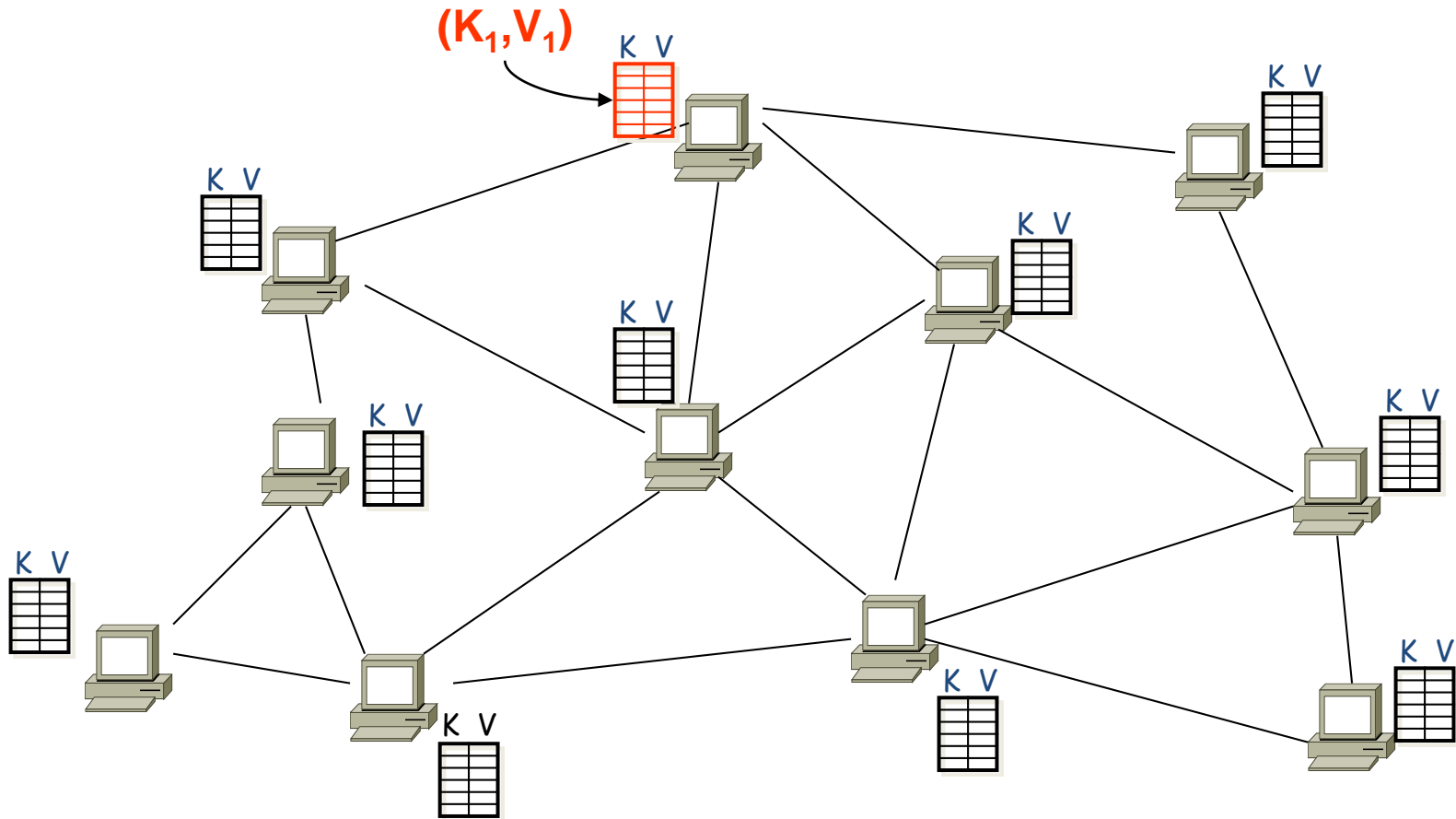
Operation: take *key* as input; route messages to node holding *key*

Cont...



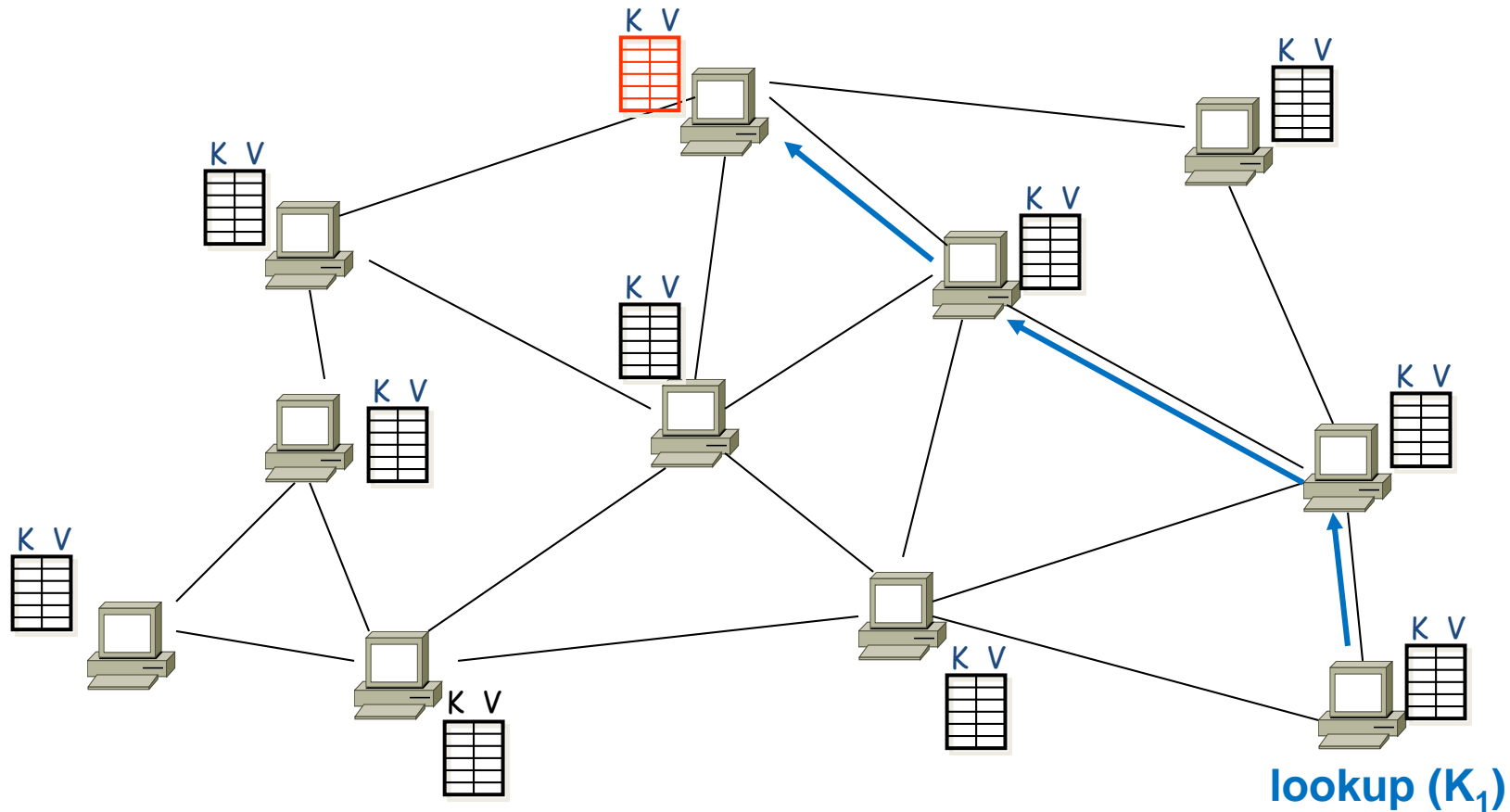
Operation: take *key* as input; route messages to node holding *key*

Cont...



Operation: take *key* as input; route messages to node holding *key*

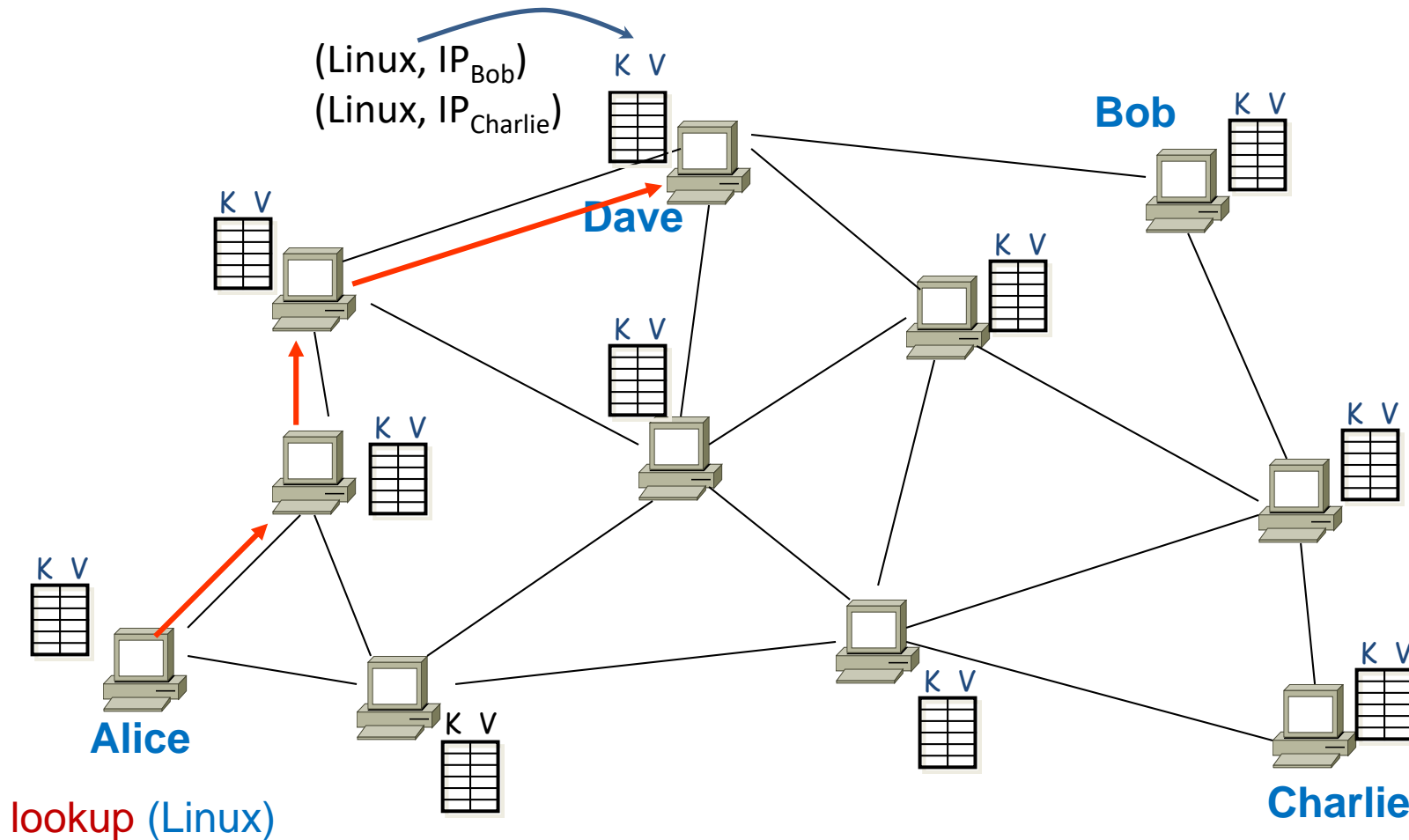
Cont...



Operation: take *key* as input; route messages to node holding *key*

DHT Service – P2P File Sharing

- “key” is the **content name**
- “value” is the **IP address** of a peer that has a copy of the content



- Naïve approach:
 - Randomly scatter the (key,value) pairs across all the peers
 - Each peer maintain a list of the IPs of all participating peers
- Querying peer sends its query to whom?
 - Answer: To all other peers
- Who will respond to the query?
 - Answer: The peers containing the (key,value) pair corresponding to the lookup 'key'
- Limitations of this design:
 - Unscalable as each peer needs to know about all other peers
 - Traffic overhead as each query is sent to all peers

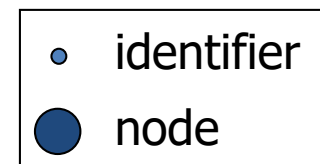
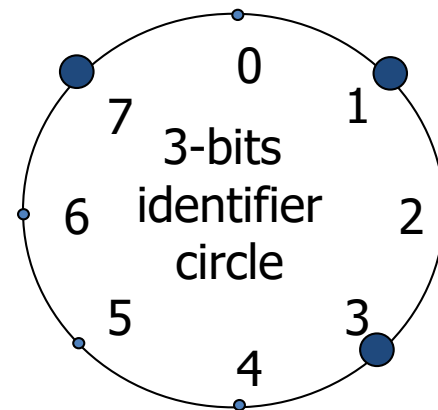
DHT design attributes



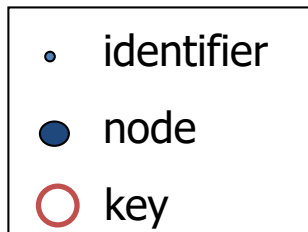
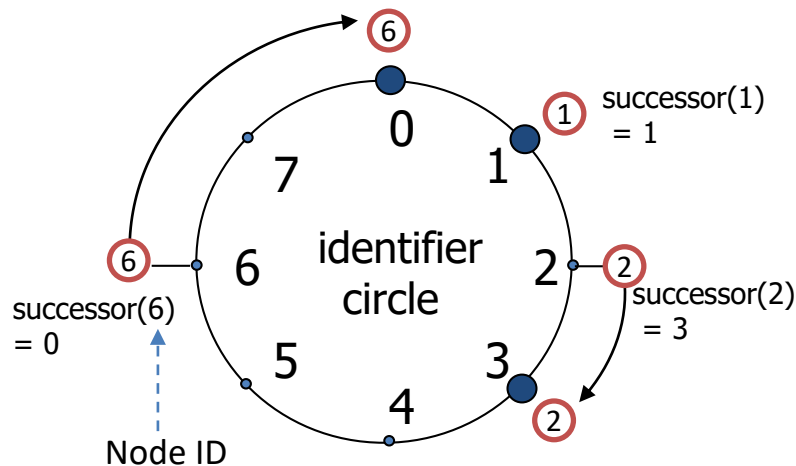
- State Assignment:
 - what “(*key*, *value*) tables” does a node store?
 - How do the “keys” are associated with “peers”?
- Network Topology:
 - how does a node select its neighbors?
- Routing Algorithm:
 - which neighbor to pick while routing to a destination?
- Various **DHT algorithms** make different choices
 - **Chord**, CAN, Pastry, Tapestry, Plaxton, Viceroy, Kademlia, Skipnet, Symphony, Koorde, Apocrypha, Land, ORDI ...

Circular DHT - Chord

- Chord is a protocol and algorithm for a P2P distributed hash table (DHT).
- Chord specifies
 - how **keys** are assigned to **nodes**,
 - how a **node** can discover the **value** for a given **key** by locating the node responsible for that key.
- It was introduced in 2001 by Stoica et al. at MIT
- **Nodes** and **keys** are assigned an **m-bit identifier** using **consistent hashing**.
 - Consistent hashing uniformly distribute both the keys and nodes in the same identifier space with a negligible possibility of collision
- The **SHA-1** algorithm is the base **hashing function** for consistent hashing.
- Using the Chord lookup protocol,
 - *nodes* and *keys* are arranged in an **identifier circle** that has at most 2^m nodes, ranging from 0 to 2^m-1 .
 - Some of these nodes will map to machines or keys, while **others are empty**.



Cont...

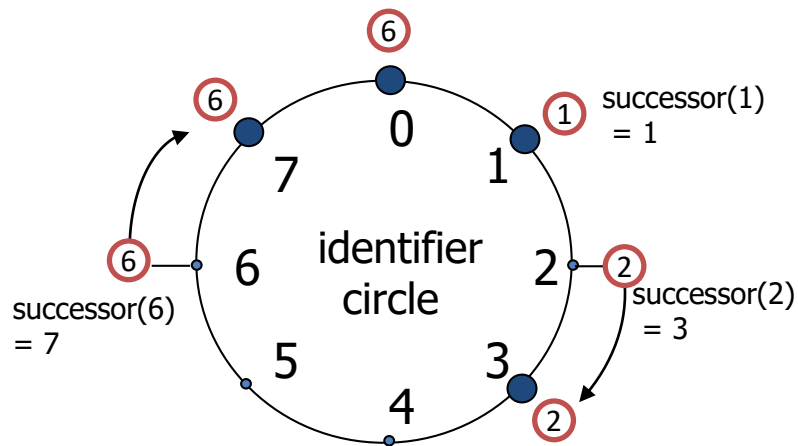


- Each node has a *successor* and a *predecessor*.
- The *successor node* of a **key** k is the first **node** whose **ID** equals to k or follows k in the identifier circle, denoted by *successor*(k)
- A key is stored at its **successor** node
- So *looking up* a **key** k is to query *successor*(k)

Home Work:

- The basic approach is to pass the query to a node's successor, if it cannot find the key locally.
- This will lead to a $O(N)$ query time where N is the number of machines in the ring.

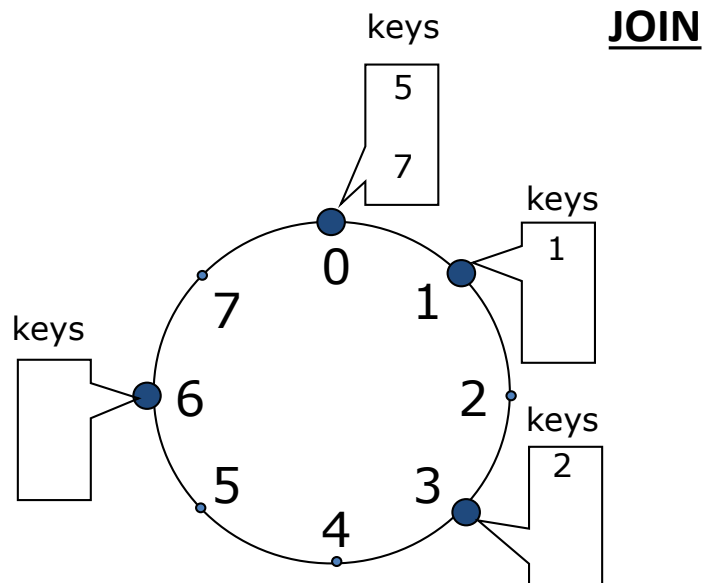
Cont...



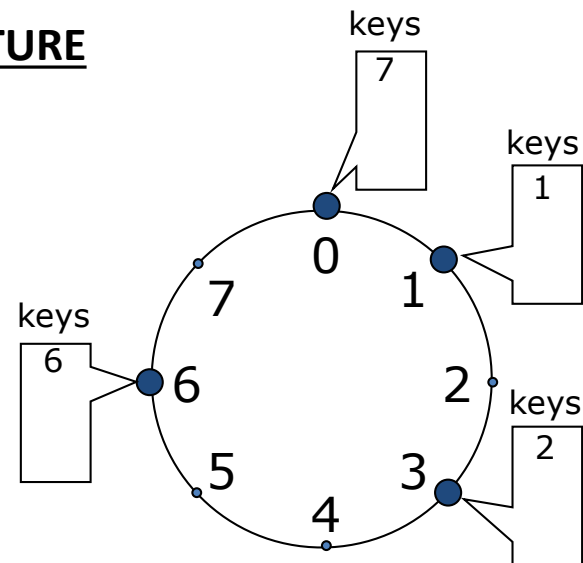
- Node **Joins** and **Departures** in the circle
- When a **node** n joins the network, certain **keys** previously assigned to n 's successor now become assigned to n .
- When **node** n leaves the network, all of its assigned **keys** are reassigned to n 's successor.
- Consistence Hashing Function **balances load**: all nodes receive roughly the same number of keys

Home Work:

- When an n -th **node** joins (or leaves) the network, only an $O(1/n)$ fraction of the **keys** are moved to a different location

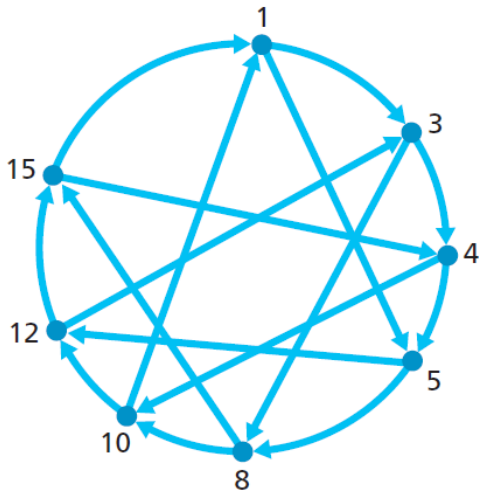


DEPARTURE



Cont...

- Although each peer is only aware of **two neighboring** peers, to find the node responsible for a key (in the worst case), all N nodes in the DHT will have to forward a message around the circle; **$N/2$ messages are sent on average**.
- Thus, in **designing a DHT**, there is **tradeoff** between
 - the **number of neighbors** each peer has to track and
 - the **number of messages** that the DHT needs to send to resolve a single query.
- we can **refine our designs** of DHTs
 - use the circular overlay as a foundation,
 - but add “**shortcuts**” so that each peer not only keeps track of its immediate successor and predecessor, but also of a relatively **small number of shortcut peers** scattered about the circle.



- **Shortcuts** are used to **expedite the routing** of query messages.
- **For example**, when **peer 4** receives the message asking about **key 11**, it determines that the closet peer to the key (among its neighbors) is its shortcut neighbor 10, and then forwards the message directly to peer 10.
- It has been shown that the DHT can be designed so that both the **number of neighbors per peer** as well as the **number of messages per query** is **$O(\log N)$** , where **N** is the number of peers.

Thanks!