5. Peer-to-peer (P2P) networks

PA191: Advanced Computer Networking I.

Eva Hladká

Slides by: Tomáš Rebok

Faculty of Informatics Masaryk University

Autumn 2020

Lecture Overview I

1. Client-Server vs. Peer-to-Peer
   - Client-Server Systems
   - P2P Systems
   - Comparison
2. Generic P2P Architecture
   - Overlays and Peer Discovery
   - Service/Resource Discovery
3. Taxonomy of P2P Systems
   - Centralized P2P Systems
   - Decentralized P2P Systems
   - Hybrid P2P Systems
4. Routing in P2P Networks
   - Introduction, Motivation
   - Routing in Unstructured P2P Networks
   - Routing in Structured P2P Networks
   - Routing in Hybrid P2P Networks
5. Information Sources

Lecture Overview I

Distributed Applications I.

- a distributed application consists of multiple software modules located on different computers
  - the modules interact with each other over a communication network connecting the different computers
    - the communication network is used for synchronisation and communication between the modules
  - it is possible that multiple users may use the application concurrently on different computers
- to build a distributed application, it is necessary to decide:
  - how to place those software modules on the different computers in the network
  - how each software module discovers the other modules it needs to communicate with

Distributed Applications II.

- two basic approaches:
  - Client-Server architecture
  - Peer-to-Peer (P2P) architecture
- hybrids are possible and indeed useful

Client-Server Architecture I.

A client-server system comprises of two types of software modules:

- *server module*
  - one centralized instance
    - but might be internally replicated for scaling purposes
  - passively listens for connections from clients
  - multiple client requests may be handled:
    - sequentially
    - concurrently (multithreaded servers)
    - by several replicated servers at different locations
  - pending clients' requests may be queued up
  - servers are assumed to be reliable, often running in a data centre (dedicated/virtualized hardware)

Client-Server Architecture II.

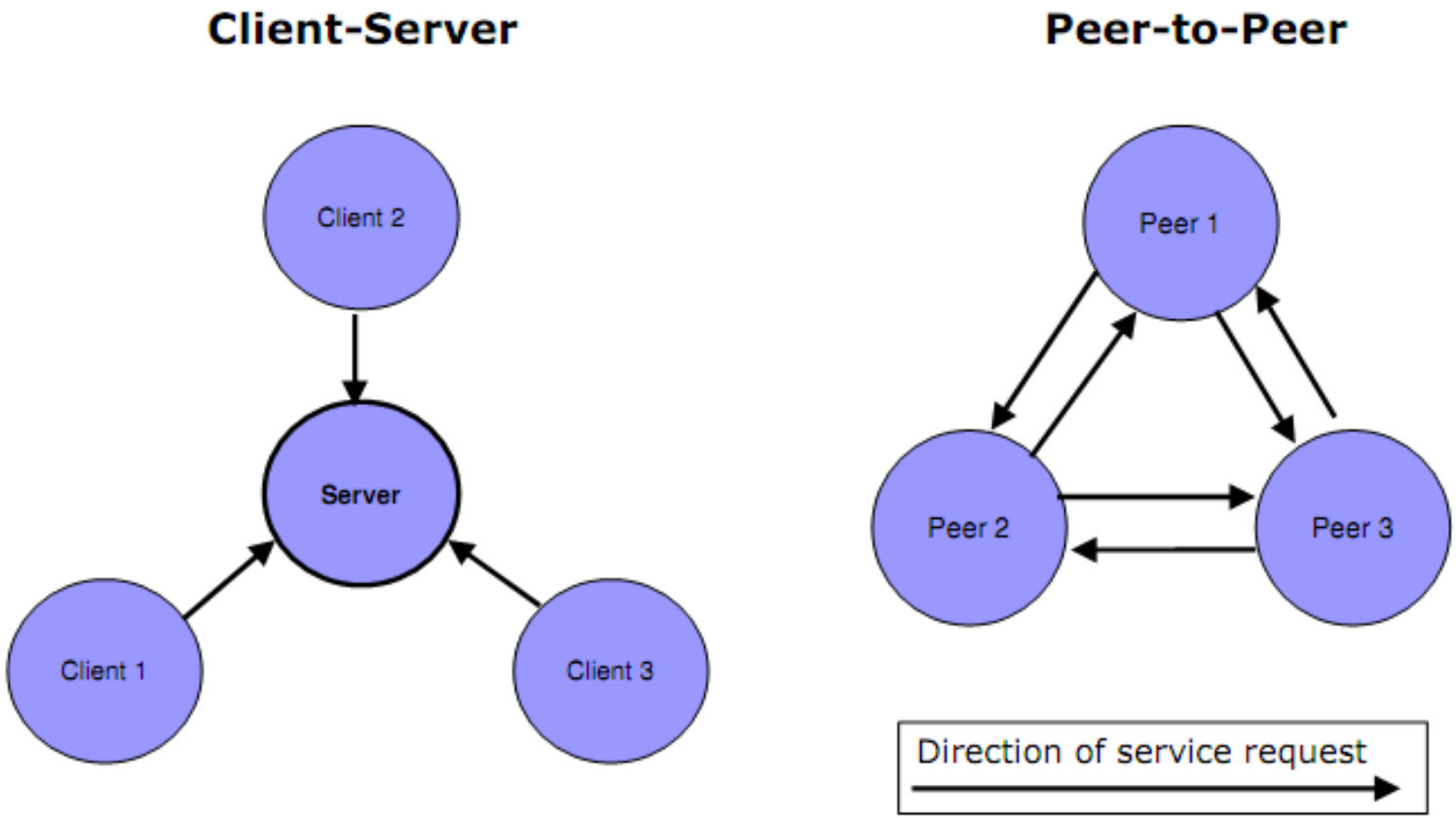
- *client module*
  - multiple distributed instances, possibly controlled by different users
  - actively initiates a connection to a server
  - no direct communication between clients
  - clients need to know the network address and port number of a server
    - service discovery is typically performed through client configuration
  - clients may be unreliable without affecting overall system stability

- examples of client-server systems:
  - web server/web browsers
  - web server/client applications (web services)
  - SSH/Telnet/FTP server/clients
  - NFS/SMB server/clients
  - ...

P2P Architecture

- a P2P system consists of many identical software modules (peers) running on different computers
- peers communicate directly with each other
- each peer is a server as well as a client:
  - provides services to other peers
  - requests services from other peers
- unlike dedicated servers, peers tend to be *unreliable*
- service discovery is more complicated since there are many servers continuously appearing and disappearing at different network locations
- provide natural scalability due to multiple servers
- can work without allocating dedicated server machinery

Communication Structure Comparison



Peer-to-Peer Systems Definition

Peer-to-Peer Systems

Peer-to-peer (P2P) systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority.

P2P Properties

- *Symmetric role*
  - each participating node typically acts both as a server and as a client
    - however, in many designs this property is relaxed by the use of special peer roles ("super peers" or "relay peers")
- *Scalability*
  - P2P systems can scale to thousands of nodes
    - the P2P protocols cannot require "all-to-all" communication or coordination
- *Heterogeneity*
  - a P2P system is (usually) heterogeneous in terms of the hardware capacity of the nodes
- *Distributed control (Decentralization)*
  - ideally, no centralized structures should exist in P2P systems
- *Dynamism*
  - the topology of P2P systems may change very fast due to joining of new nodes or leaving existing ones
- *Resource sharing*
  - each peer contributes system resources (computing power, data, bandwidth, presence, etc.) to the operation of the P2P system
- *Self-organization*
  - the organization of the P2P system increases over time using local knowledge

P2P Applications

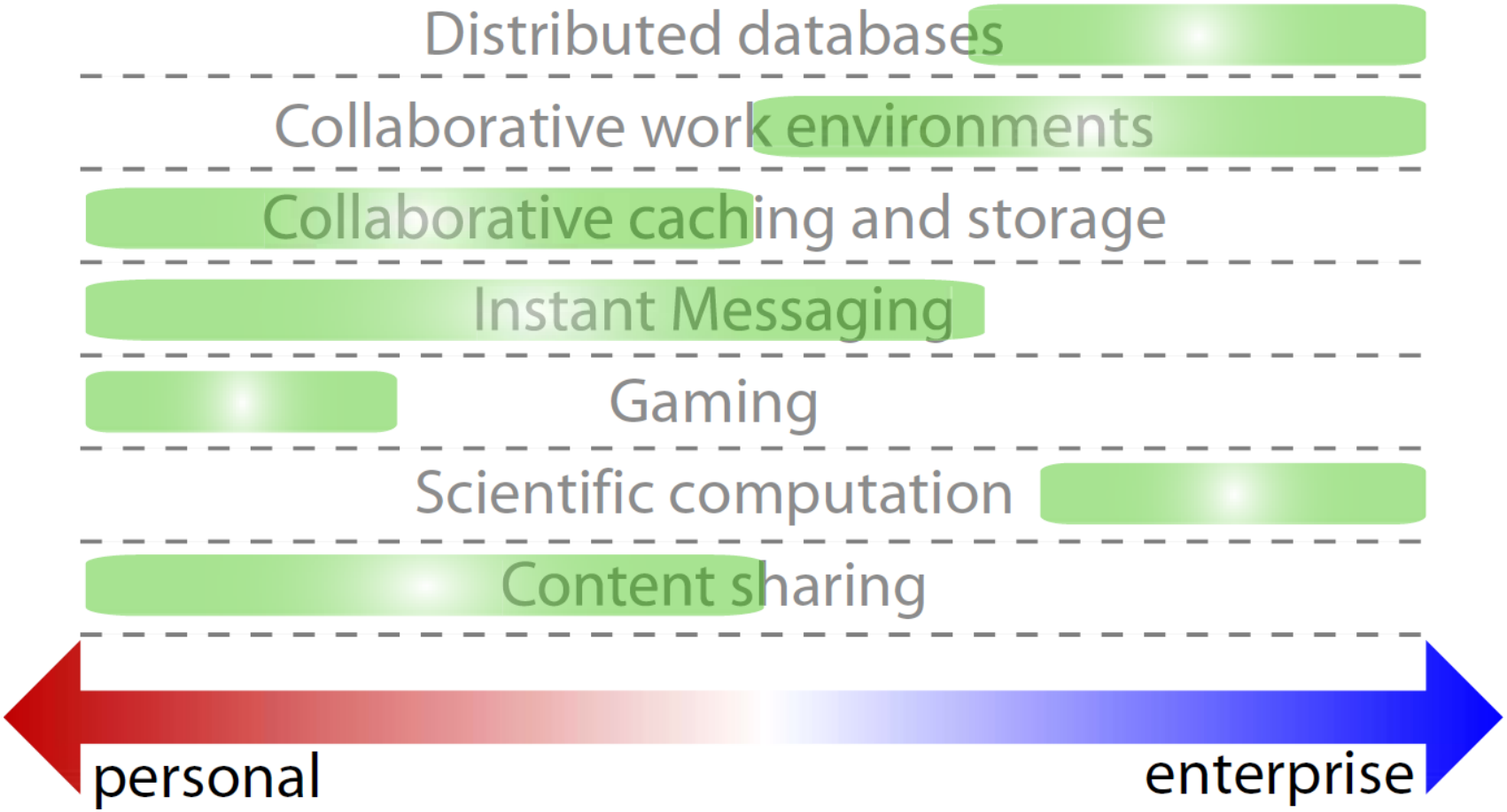


Figure: P2P Applications.

Client-Server vs. Peer-to-peer

Comparison I.

The systems can be compared from several points of view:

- *Ease of development*
  - C-S is more established and familiar than P2P
  - C-S exhibits simple interaction patterns for clients and server, while P2P involves more complex interaction patterns between peers
- *Manageability*
  - it is easier to maintain a centralized server in a C-S environment than keeping a track of and maintaining several distributed peers in a P2P system
- *Scalability*
  - C-S scalability is limited by fixed server hardware, though scaling can be achieved through load balancing over multiple servers at increased cost
  - P2P is scalable by nature, since as the number of peers grows, so does the "server" capacity

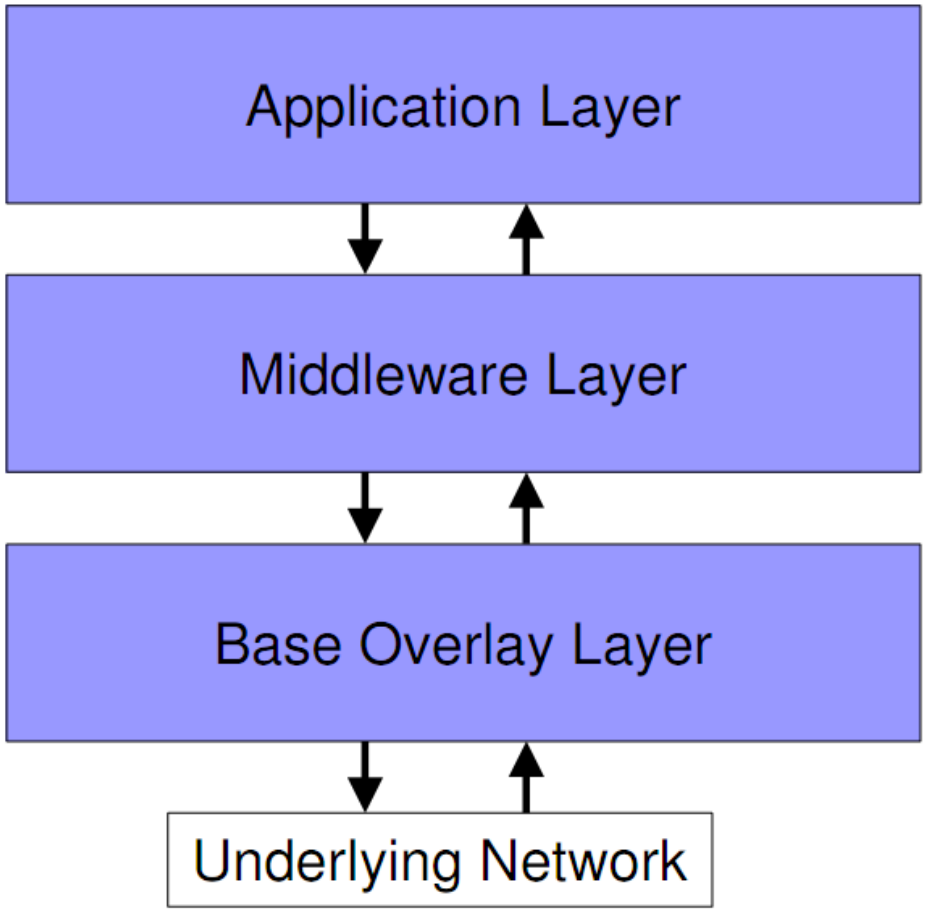
Client-Server vs. Peer-to-peer

Comparison II.

- *Security*
  - responsibility for the C-S security lies within the server, which is centrally hosted in a secure environment
  - responsibility for P2P security is distributed across peers in different administrative domains, some of which might be compromised
- *Reliability*
  - the C-S's reliability is achieved through the use of multiple redundant servers (possibly hosted at different locations) with automatic fail-over, at additional cost
  - with P2P, resilience comes free of charge, since multiple peers are usually able to provide the same service in the case that some peers fail

Lecture Overview I

P2P Architecture



- libraries exist that provide reusable P2P functionality (e.g. JXTA)
- some applications integrate all of the above (e.g., Gnutella, Bittorrent, etc.)

P2P Architecture

Base Overlay Layer I.

- the *base overlay layer* is responsible for:
  - discovering new peers
  - maintaining the P2P overlay (virtual) network
  - forwarding messages between peers
- the *overlay network* is a virtual network laid over the "physical" network (e.g. TCP/IP)
  - overlay network "wires" are implemented using underlying network facilities (e.g. TCP connections or UDP messages)
  - overlay network distance is measured in the number of hops from peer to peer
    - peers, that are distant in the physical network may be neighbours in the overlay network, and vice-versa
  - the performance of the P2P system is influenced by the structure of the overlay network

P2P Architecture

Base Overlay Layer II.

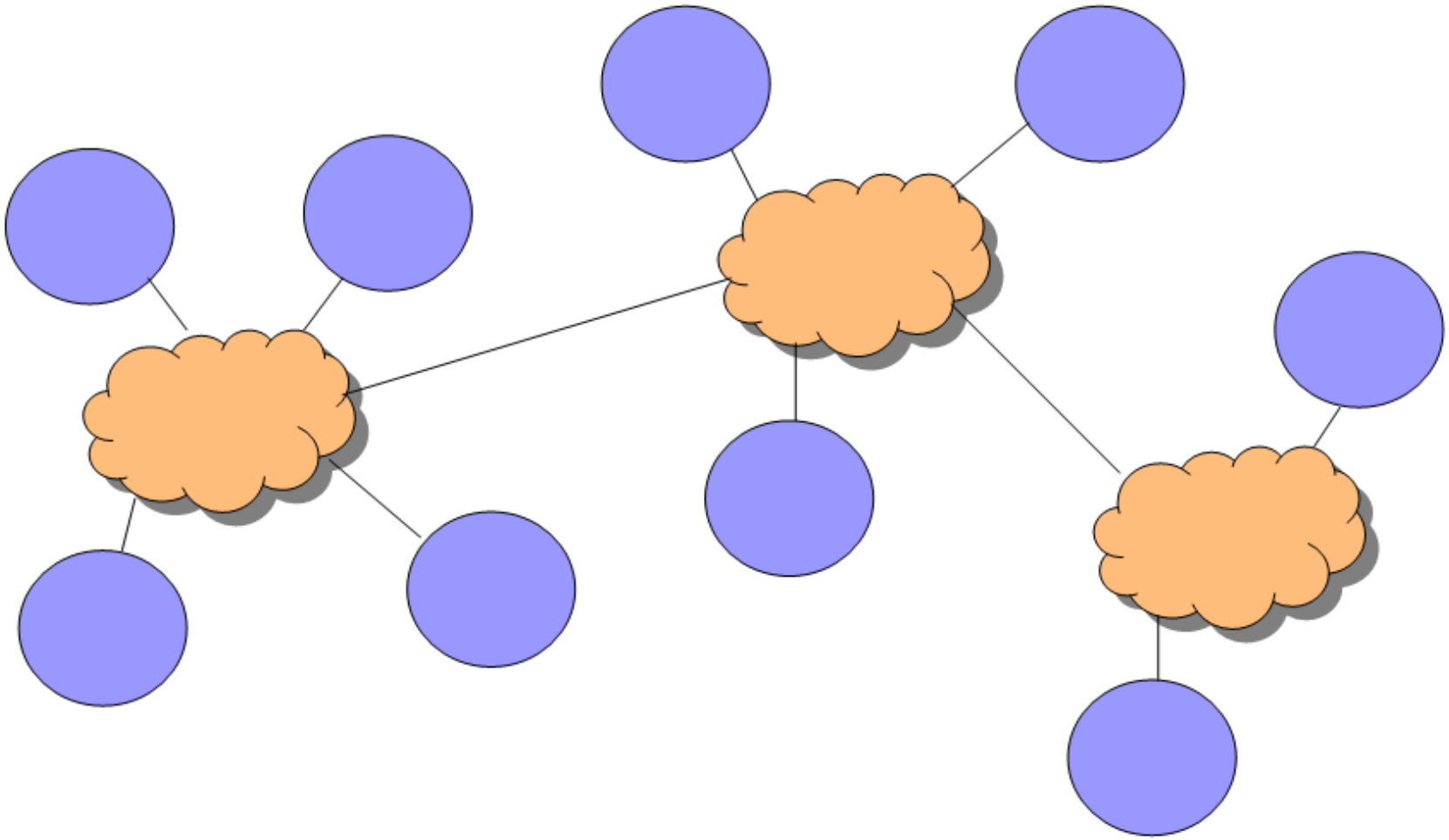


Figure: Overlay vs. Underlying Network.

P2P Architecture

Base Overlay Layer II.

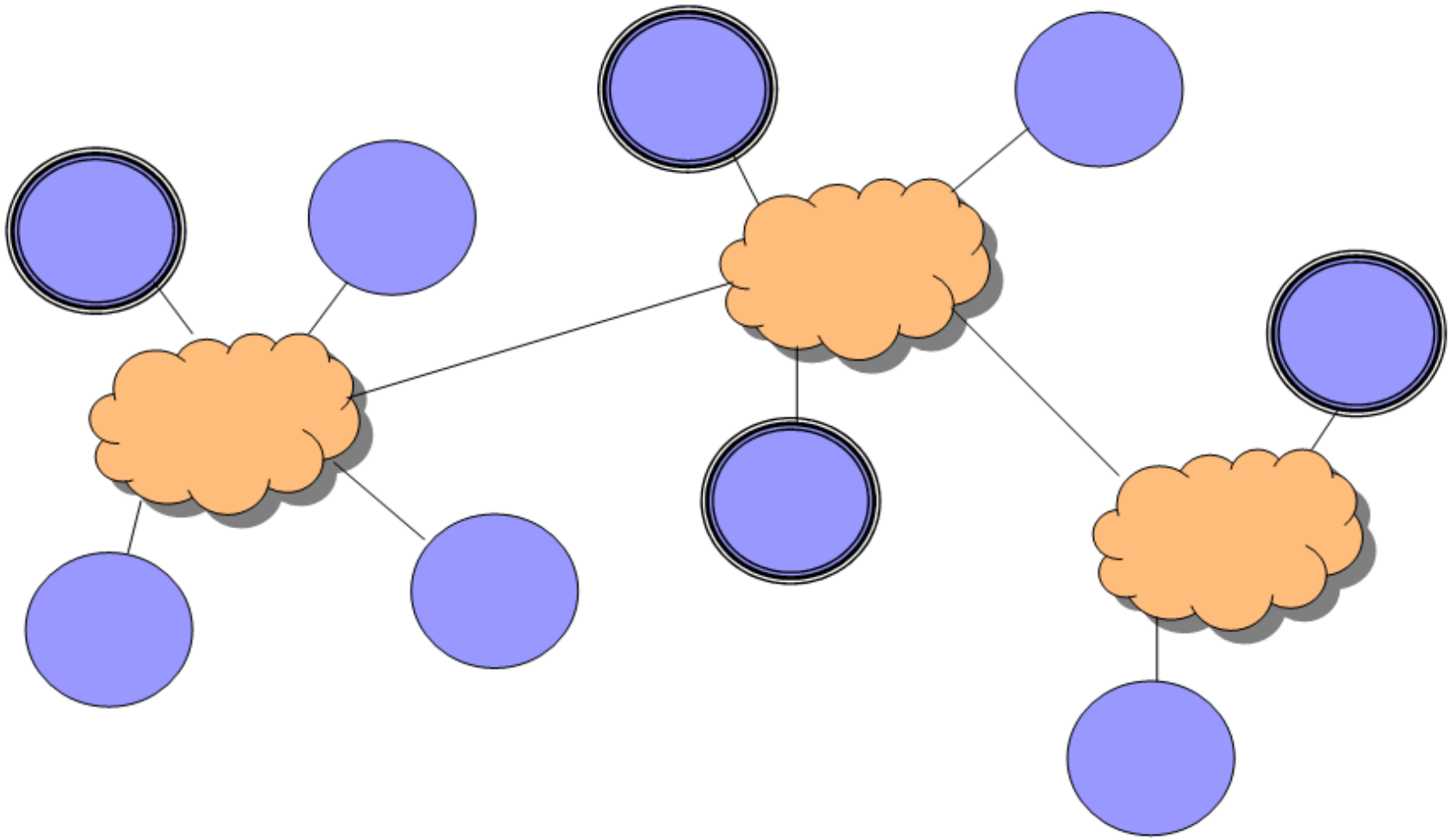


Figure: Overlay vs. Underlying Network.

P2P Architecture

Base Overlay Layer II.

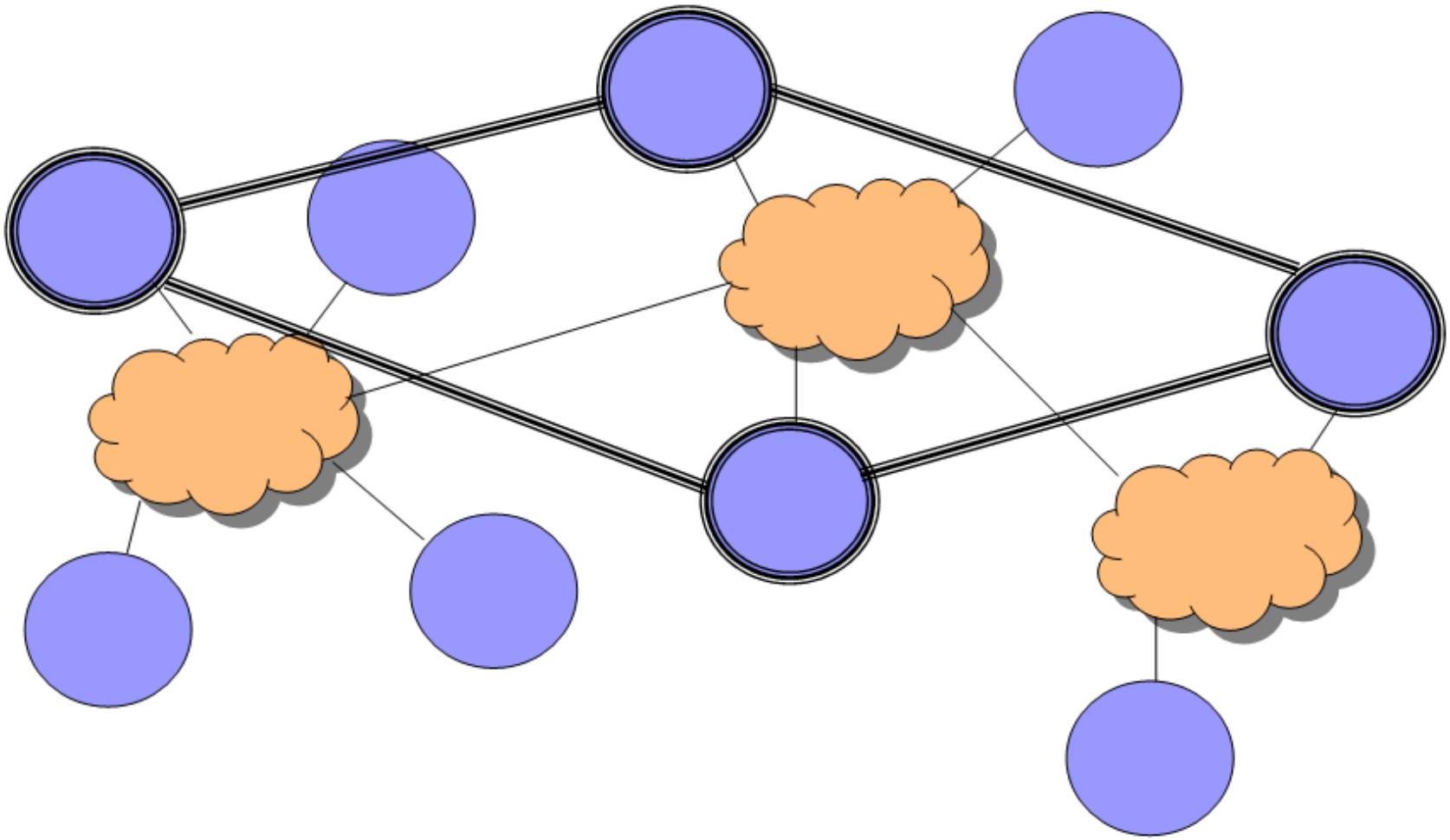


Figure: Overlay vs. Underlying Network.

P2P Architecture

Base Overlay Layer II.

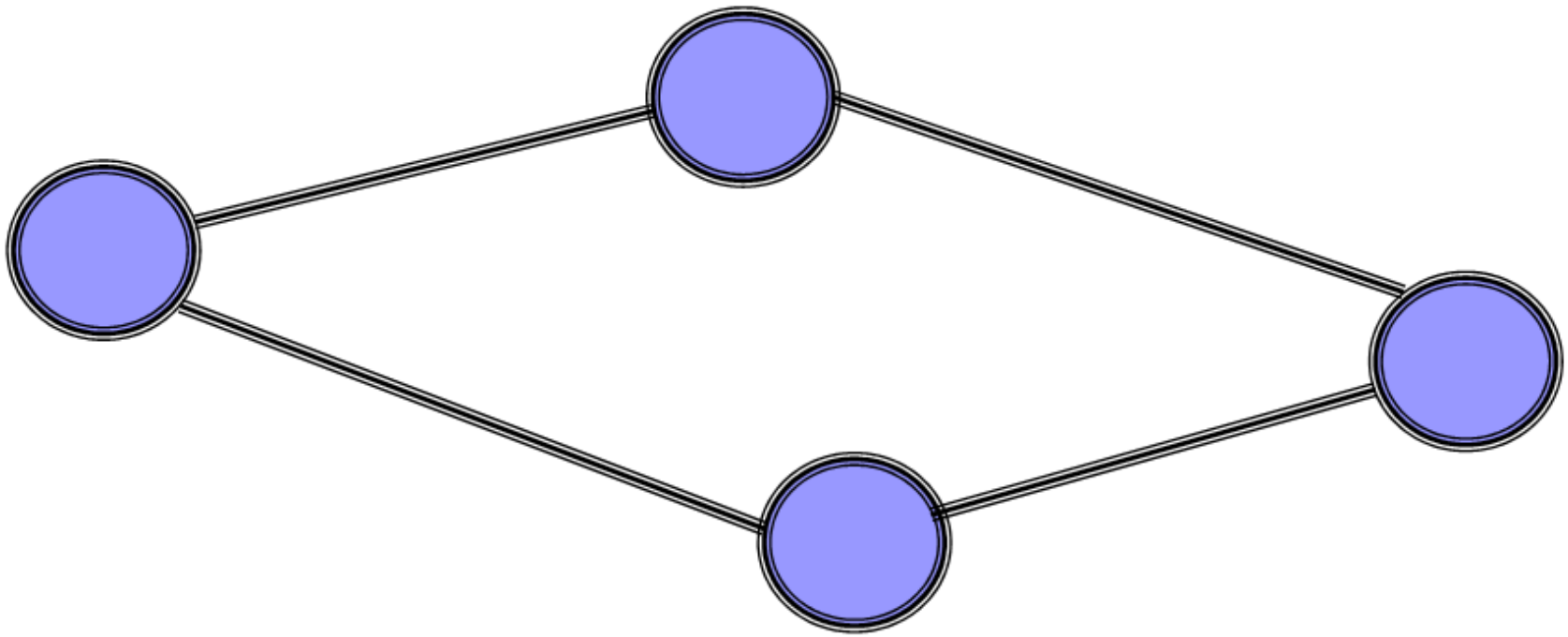
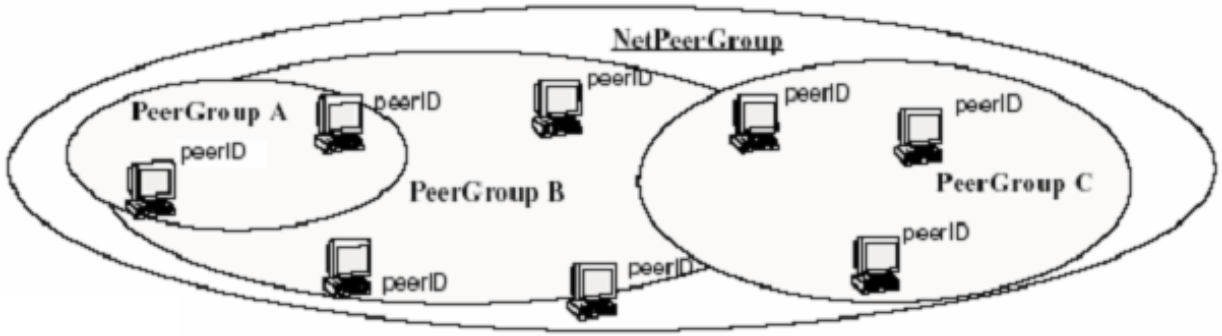


Figure: Overlay vs. Underlying Network.

P2P Architecture

Middleware Layer

- the *middleware layer* facilitates P2P application development by hiding overlay and service discovery issues
- it provides access to the services/resources provided by peers, and may be responsible for functions such as:
  - **security:** controlling access to services/ resources
  - **service/resource discovery:** searching and indexing services/resources distributed across peers
  - **peer groups:** coordinating peers that provide or consume a particular service/resource
    - may provide fault tolerance and persistent state



- e.g., JXTA (Java P2P platform), BOINC, P2, Windows P2P Networking,

P2P Architecture

Application Layer

- the middleware services can be used to build complete applications:
  - file sharing – e.g., Napster, Gnutella, Kazaa, ...
  - routing protocols
  - instant messaging, videoconferencing applications – e.g., Skype
  - distributed file systems
  - distributed backup systems
  - distributed computing – e.g., grid computing, SETI@Home, ...
  - and many many more...

Overlays and Peer Discovery

- a P2P network is typically a "virtual" network overlaid on an existing network (e.g. the Internet)
  - the overlay is used for indexing and peer discovery and make the P2P system independent from the physical network topology
  - content is typically exchanged directly over the underlying IP network
- a new peer needs to discover at least one existing peer in order to join a P2P network
  - network location information: IP address, listening port number, etc.
- if no peers are found immediately, the new peer either
  - passively waits for new participants, or
  - proactively looks for potential new participants
- it is hard to locate existing peers in a large network such as the Internet

Overlays and Peer Discovery

Initial Peer Discovery I.

Static configuration:

- each peer is preconfigured with a list of the network locations (IP address and port number) of every other peer in the system
- on startup (and possibly periodically) each peer attempts to connect to some other peers in its list, some of which may be running
- due to the manual configuration, this is only suitable for P2P networks with a small number of peers which do not change frequently
- can alternatively be used to initially contact a small number of "well-known" peers that are guaranteed to be online

Overlays and Peer Discovery

Initial Peer Discovery II.

Centralized directory:

- each peer is preconfigured with the network location of a centralized server
- each peer contacts the server on startup (and possibly periodically) to:
  - obtain an updated list of currently active peers
  - indicate to the server that it is active
- most subsequent communications bypass the server, using the P2P overlay network to route messages instead
  - occasionally, other services are also provided by the server (e.g. a list of files hosted by each peer)
- peers may go offline
  - cleanly, the peer's shutdown procedure contacts the server to remove it from the active peer list
  - without warning (crash, network or power failure), making the server's active peer list obsolete (it's necessary to use active peer list item expiry and periodic liveness checks)
- usually, a peer only needs to connect to a few peers on the overlay network
  - the other members can be discovered by the *member propagation techniques*
- centralized directory server is a single point of failure

Overlays and Peer Discovery

Initial Peer Discovery III.

Member Propagation Techniques with Initial Member Discovery:

- in general, it is not necessary to discover all of the participating members in the network
  - in many cases, discovering a subset of the participating members is adequate
- after discovering just one existing peer, information about the rest of the P2P network can be obtained from it
  - if each peer maintains a full member list $\rightarrow$ easy for any new peer to obtain a full member list from any other peer
  - alternatively, each peer can maintain a partial member list, replacing offline peers with new ones from neighbouring peers' lists

Overlays and Peer Discovery – Overlay Network Topology

- intermediate peers in the overlay network forward messages between indirectly connected peers
- the overlay topology significantly affects P2P system performance
- two key properties determine the effectiveness of the overlay mesh:
  - **Diameter:** longest distance between any two peers (overlay hops or latency)
    - should be minimized
  - **Average Degree:** average number of links per peer (high AD increases message load, but improves fault tolerance)
    - should be kept at a moderate level
- it is necessary to avoid linear formations and splits in the mesh
- common topologies:
  - *Random Mesh*
  - *Tiered*
  - *Ordered Lattice*

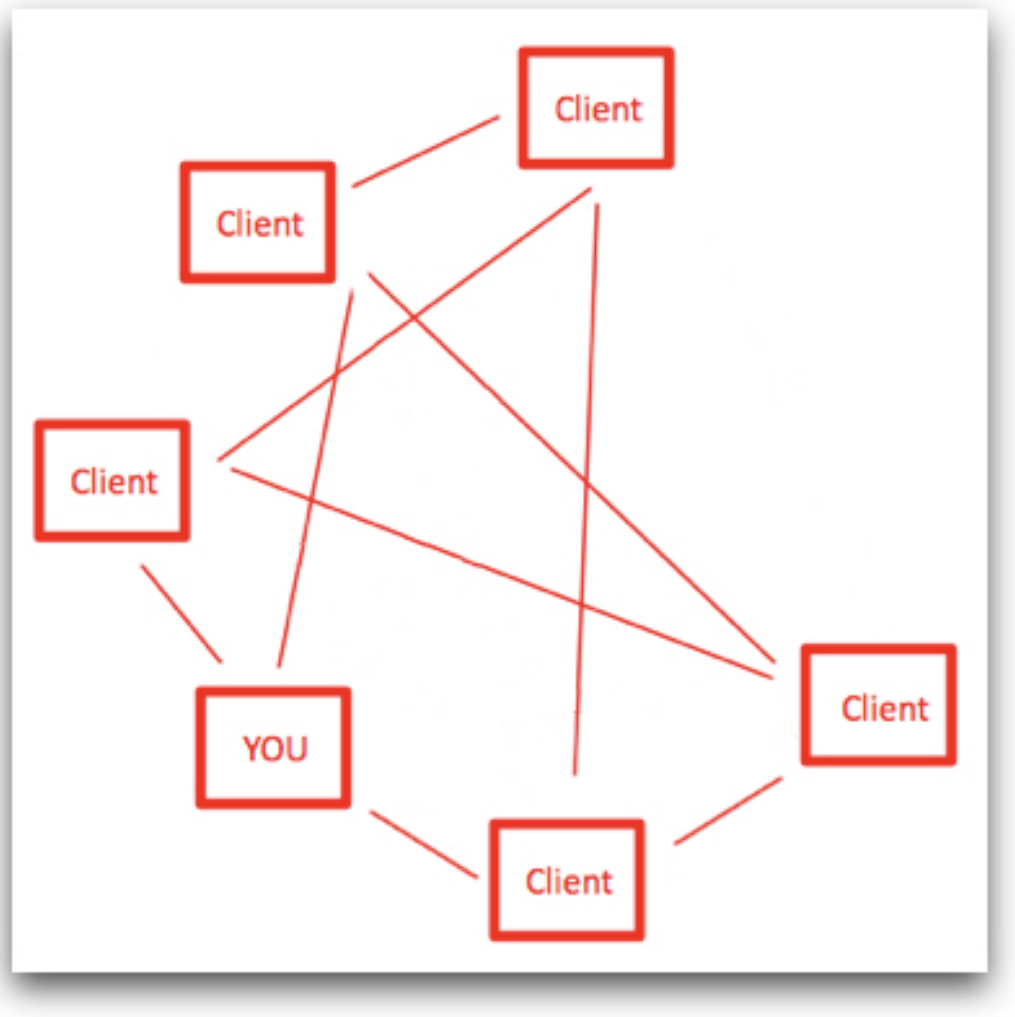
Overlays and Peer Discovery – Overlay Network Topology

Random Mesh

- each peer discovers a number of other peers and attempts to connect to them indiscriminately
- this (hopefully) results in a random structure with uniform degree
- distant peers on underlying network could be overlay neighbours
  - *solution:* connect to peers with lowest latency
- random mesh is suitable for linking a large number of peers with uniform resources and connectivity
- search message flooding can easily be used to discover resources/services on other peers
  - but generates a lot of traffic

Overlays and Peer Discovery – Overlay Network Topology

Random Mesh



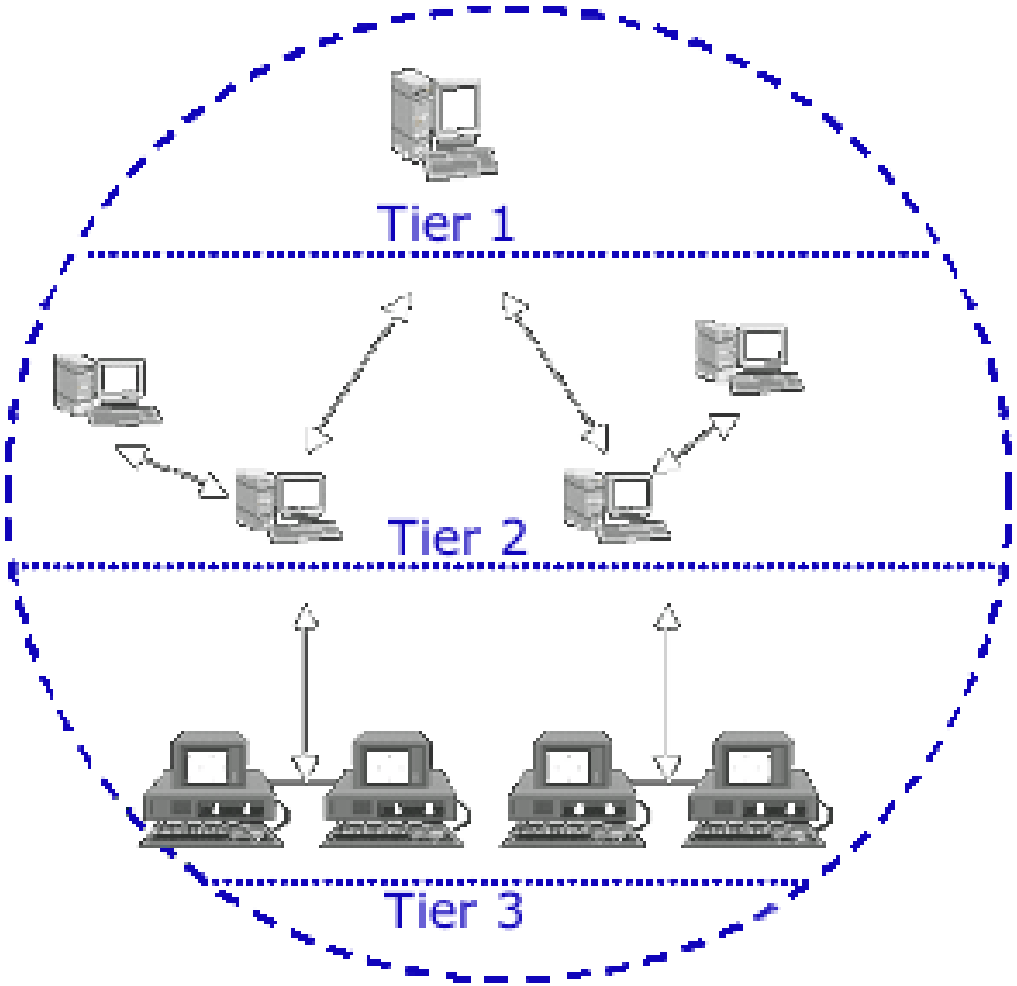
Overlays and Peer Discovery – Overlay Network Topology

Tiered Structure

- peers are ordered into tiers of a tree depending on their advertised resources and connectivity (e.g. Kazaa's nodes and supernodes, 2-tier)
  - tier 0 is the foundation tier containing (possibly well-known) reliable peers with adequate resources and message forwarding capacity
  - at each tier, every peer is linked to a number of peers of a lower tier and forwards messages up and down
  - poorly-resourced leaf peers only link to their 'super-peer' and do not forward other peers' messages; they are omitted from peer discovery
- the system needs to recover from peers leaving abruptly and disrupting the tree structure
- the hierarchy may be optimized to follow the underlying network's structure (e.g. P2P video streaming)

Overlays and Peer Discovery – Overlay Network Topology

Tiered Structure



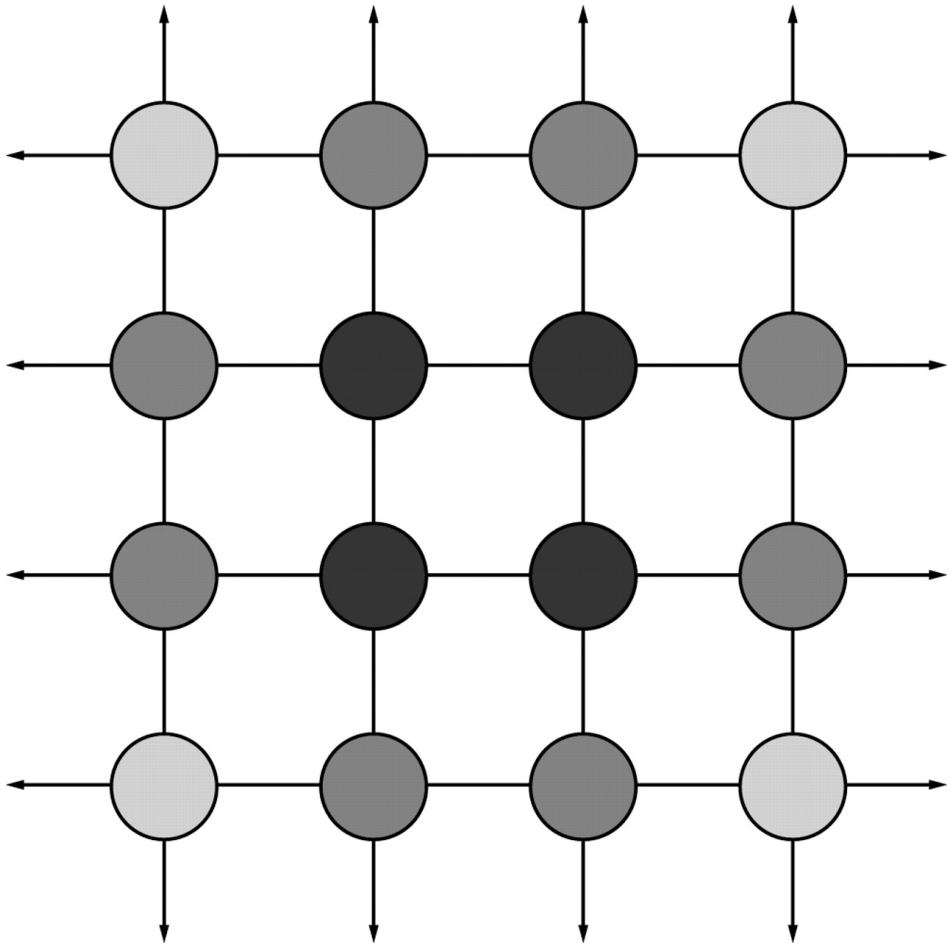
Overlays and Peer Discovery – Overlay Network Topology

Ordered Lattice

- in a two dimensional lattice, peers organize themselves in a rectangular grid:
  - each node maintains direct connections to 4 neighboring peers (except edge peers)
    - peers on opposite edges can also link to form a torus
  - can be extended to n dimensions
- messages are routed parallel to the lattice axes
- peer additions and deletions must be handled on the fly, possibly distorting the structure
  - insertions and deletions of nodes imply that different rows/columns have different numbers of members between themselves
- peer coordinates in a multi-dimensional lattice may be used as a key to locate resources in *content addressable networks (CAN)*
  - sometimes also denoted as *Distributed Hash Table (DHT)*

Overlays and Peer Discovery – Overlay Network Topology

Ordered Lattice



Service/Resource Discovery

- a peer must advertise its services to enable their discovery and subsequent use by other peers
  - e.g., in file sharing applications, the "service" is a shared file/block
- service discovery is itself a service
  - centralized – a server is asked for service location
    - Napster, UDDI for web services
  - pure P2P – a request is flooded or hashed through the peers
    - flooding, overlay multicast, CAN/DHT
- when a search message reaches a matching advertisement on a peer, the server's location is returned to the originator
- actual service messages are either routed through the overlay or directly via underlying network by the application
- can be optimized by caching advertisements/data (e.g. file/block) along search/return path on the overlay

Lecture Overview I

Taxonomy of P2P Systems I.

Generally, P2P systems can be divided into two main categories:

- **centralized** – one or more central servers are available providing various services
- **decentralized** – no central servers are employed
  - they have to consider two main design issues:
    - *the structure* – flat (single tier) vs. hierarchical (multitier)
    - *the overlay topology* – unstructured vs. structured
- besides these two, **hybrid** P2P systems also exist
  - they combine both centralized and decentralized approach to leverage the advantages of both architectures

Taxonomy of P2P Systems II.

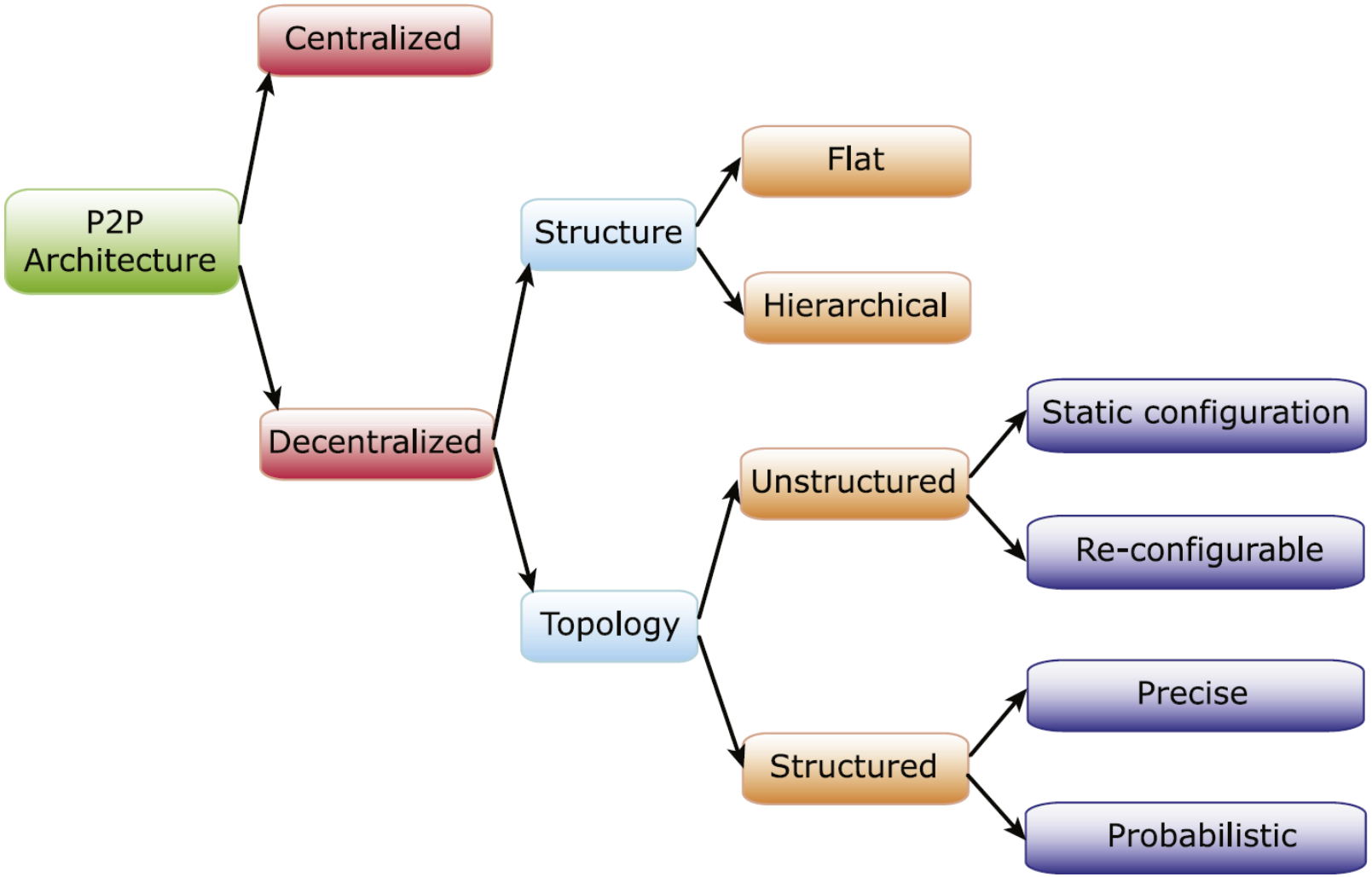


Figure: A taxonomy of P2P systems.

Taxonomy of P2P Systems III.

Centralized P2P Systems I.

Centralized P2P Systems

- combine the features of centralized (client-server) and decentralized systems
  - like a centralized system, there are one or more central servers, which help peers to locate their desired resources or act as task scheduler to coordinate actions among them
    - a peer sends messages to the central server to determine the addresses of peers that contain the desired resources
  - like a decentralized system, once a peer has its information/data, it can communicate directly with other peers
    - i.e., without going through the server anymore
- *drawbacks:*
  - susceptible to malicious attacks and single point of failure
  - a bottleneck for a large number of peers (performance degradation)
  - lacks scalability and robustness
- *examples:*
  - scientific computation – SETI@home, BOINC, Folding@home, Genome@home
  - digital content sharing – Napster, Openext
  - others – Jabber (IM), Net-Z and StarCraft (entertainment), etc.

Taxonomy of P2P Systems III.

Centralized P2P Systems II.

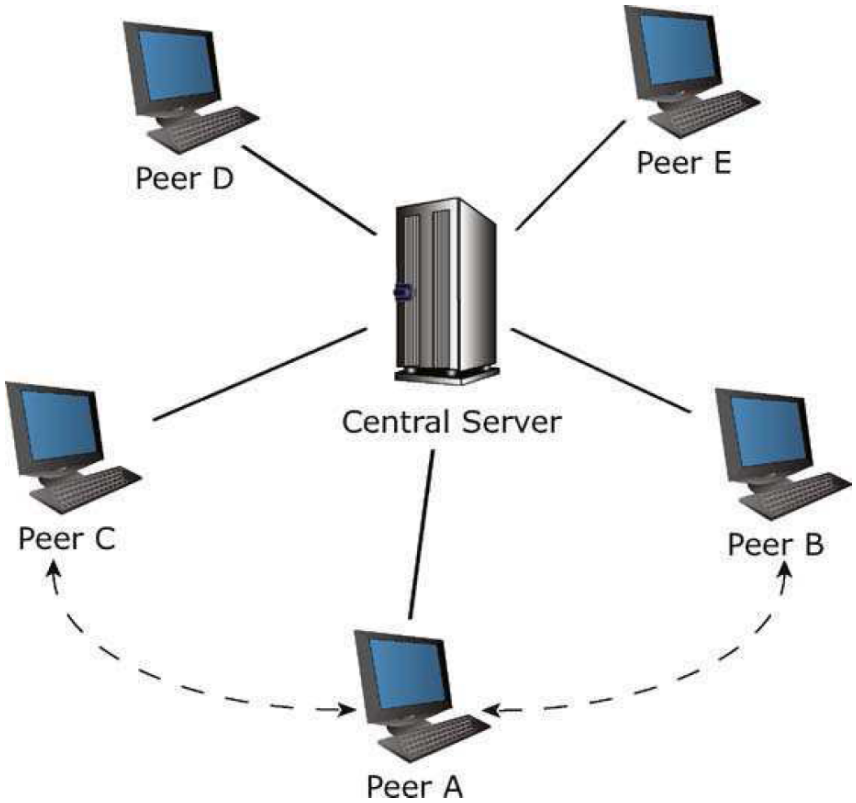


Figure: *Centralized P2P Systems:* Peer A submits a request to the central server to acquire a list of nodes that satisfy the request. Once it obtains the list (which contains Peers B and C), it communicates directly with them.

Taxonomy of P2P Systems III.

Decentralized P2P Systems I.

Decentralized (Pure) P2P Systems

- peers have equal rights and responsibilities
  - each peer has only a partial view of the P2P network and offers data/services that may be relevant to only some queries/peers
  - $\Rightarrow$ locating peers offering services/data quickly is a critical and challenging issue
- *advantages:*
  - immune to single point of failure
  - (usually) provide high performance, scalability, robustness, and other desirable features
- *examples:* Gnutella, Crescendo, PAST, FreeNet, Canon, etc.

Taxonomy of P2P Systems III.

Decentralized P2P Systems II.

Two dimensions in the design of decentralized P2P systems:

- **flat (single-tier)** vs. **hierarchical (multi-tier)** network structure
  - *flat structure* $\rightarrow$ the functionality and load are uniformly distributed among the participating nodes
  - *hierarchical structure* $\rightarrow$ multiple layers of routing structures
    - example: national level (interconnecting states), states level (interconnecting universities), universities level (interconnecting departments), etc.
    - offers certain advantages (fault isolation and security, effective caching and bandwidth utilization, hierarchical storage, etc.)

Taxonomy of P2P Systems III.

Decentralized P2P Systems II.

- **structured** vs. **unstructured** logical topology
  - *unstructured P2P system* → each peer is responsible for its own data, and keeps track of a set of neighbors that it may forward queries to
    - no strict mapping between the identifiers of objects and those of peers
    - ⇒ locating data is a challenge (its difficult to precisely predict which peers maintain the queried data)
    - ⇒ there is no guarantee on the completeness of answers (unless the entire network is searched)
    - ⇒ there is no guarantee on response time (except for the worst case where the entire network is searched)
  - *structured P2P system* → data placement is under the control of certain predefined strategies (generally, a *distributed hash table – DHT*)
    - there is a mapping between data and peers
    - ⇒ these systems can provide a guarantee (precise or probabilistic) on search cost
    - ⇒ however, typically at the expense of maintaining certain additional information
  - (systems employing a mix between structured and unstructured topology also exist)

Taxonomy of P2P Systems III.

Decentralized P2P Systems III.

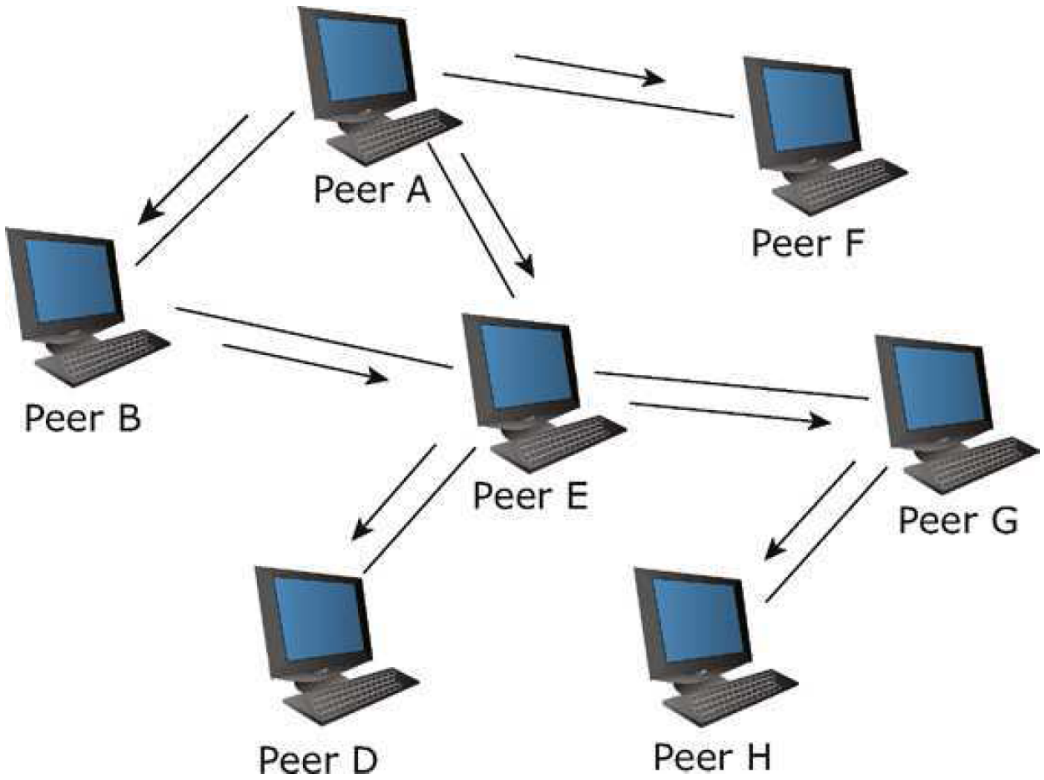


Figure: *Decentralized P2P Systems:* Peer A requests for some data that Peer D and Peer H have. The query will be broadcasted to the neighbors of Peer A, and gradually, to the other peers in the whole network (Gnutella).

Taxonomy of P2P Systems III.

Hybrid P2P Systems

Hybrid P2P Systems

- the main advantage of centralized P2P systems: quick and reliable resource locating
  - BUT with the limitation in terms of scalability
- the main advantage of decentralized P2P systems: scalability
  - BUT with the limitation in terms of longer time necessary for resource locating
- $\Rightarrow$ *Hybrid P2P systems*:
  - to maintain the scalability, there are no central servers
  - however, more powerful peer nodes are selected to act as servers to serve others
    - = *super peers*
  - $\Rightarrow$ resource locating can be done by both decentralized and centralized search techniques (asking super peers)

Taxonomy of P2P Systems III.

Hybrid P2P Systems III.

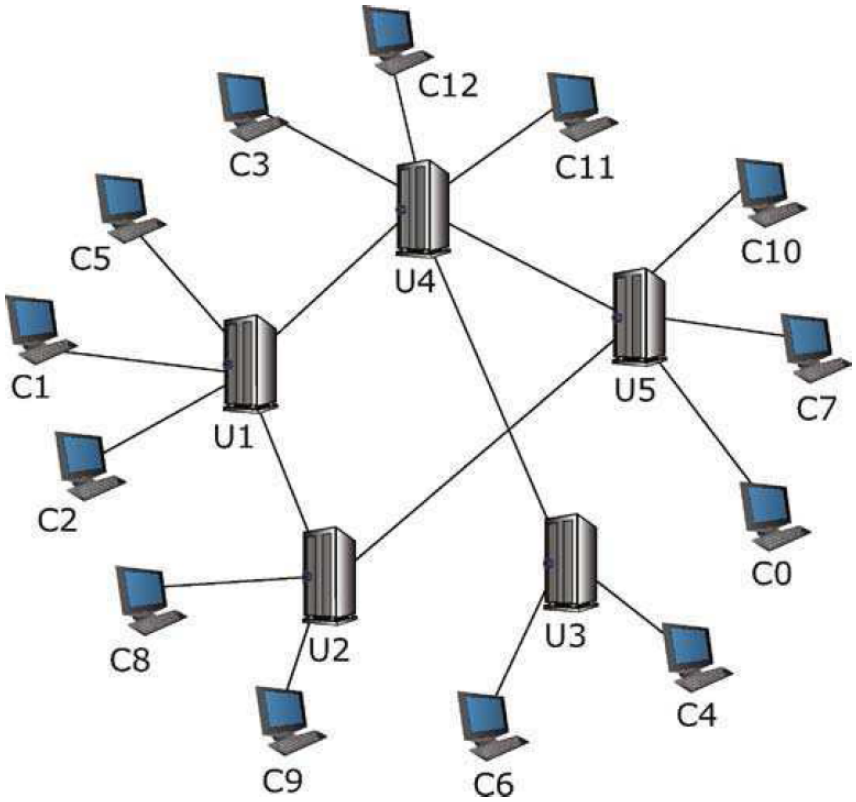
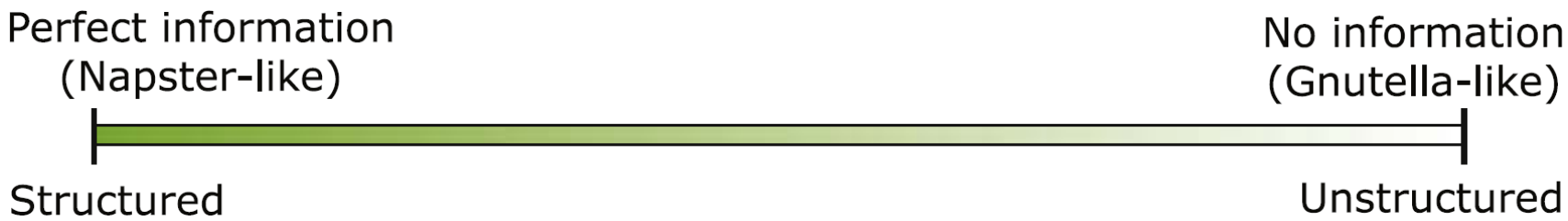


Figure: *Hybrid P2P Systems:* At first, the query is forwarded to a *superpeer/ultrapper* node, which the query node belongs to; the superpeer in cooperation with other superpeers looks for the superpeer (lookup based on a structured algorithm), which maintains a node having an answer for the request.

Lecture Overview I

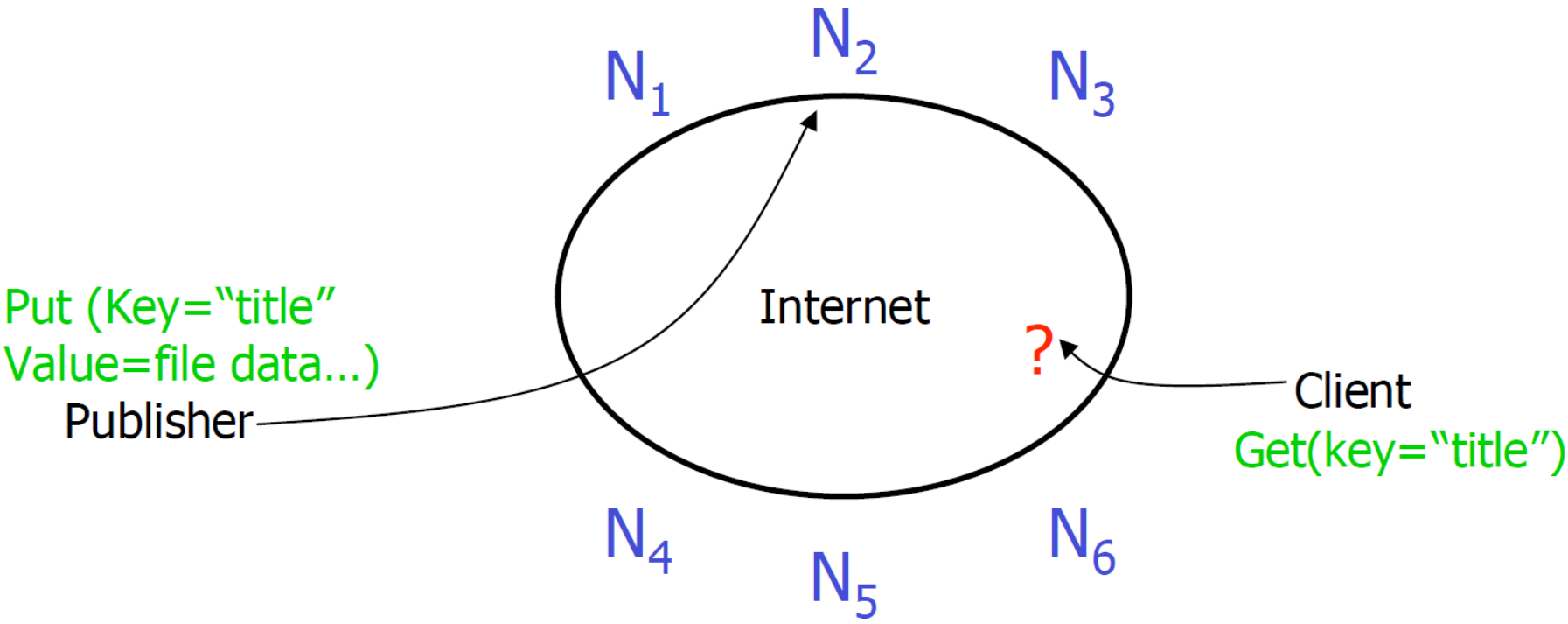
Routing in P2P Networks

- routing of messages/requests is one of the key operations in P2P systems
  - to locate desired resources, each peer should be able to forward queries to a subset of neighbor peers that are closer to the destination than any other peer
  - $\rightarrow$ the design of routing protocols is one of the most widely researched issues
- the key differences between the various schemes lie in the amount of information (metadata) being maintained at each peer
  - and how this information is organized
  - no metadata $\Rightarrow$ there is no other way for locating information except for flooding/broadcasting the request through the network



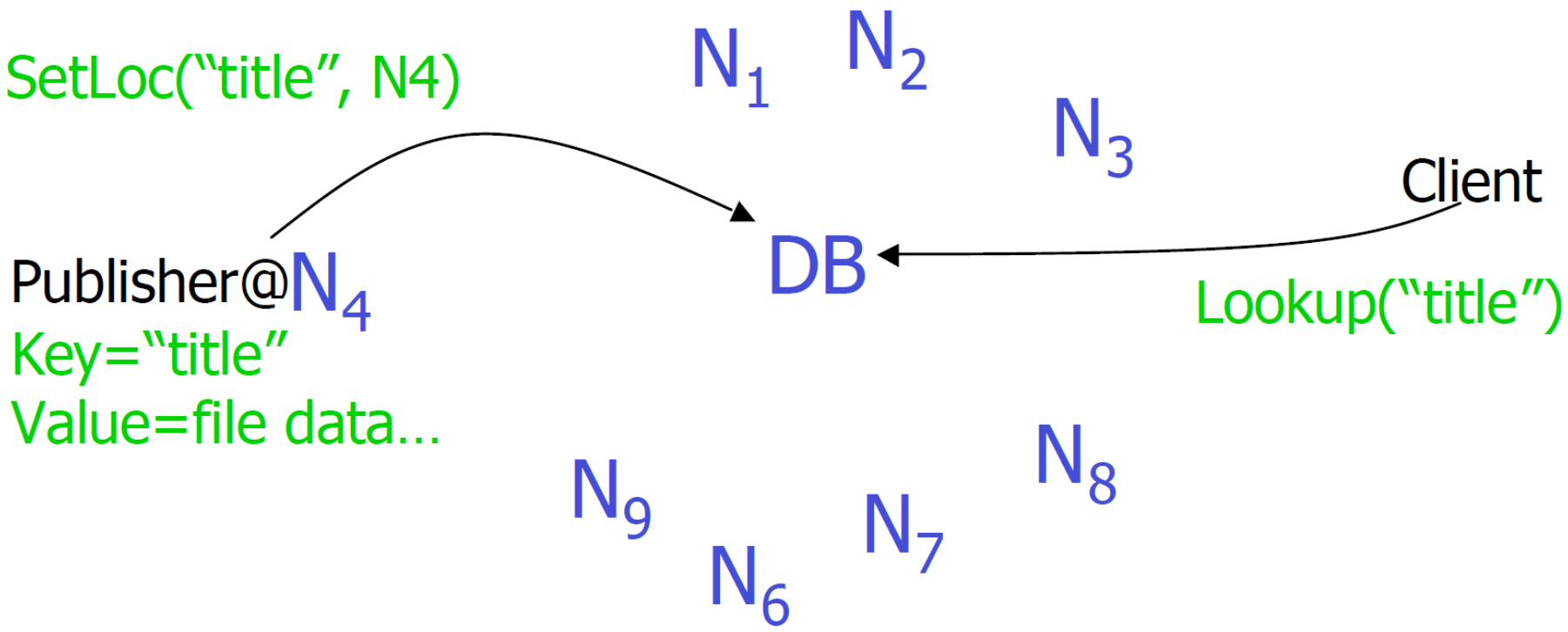
Routing in P2P Networks

The Lookup Problem



Routing in P2P Networks

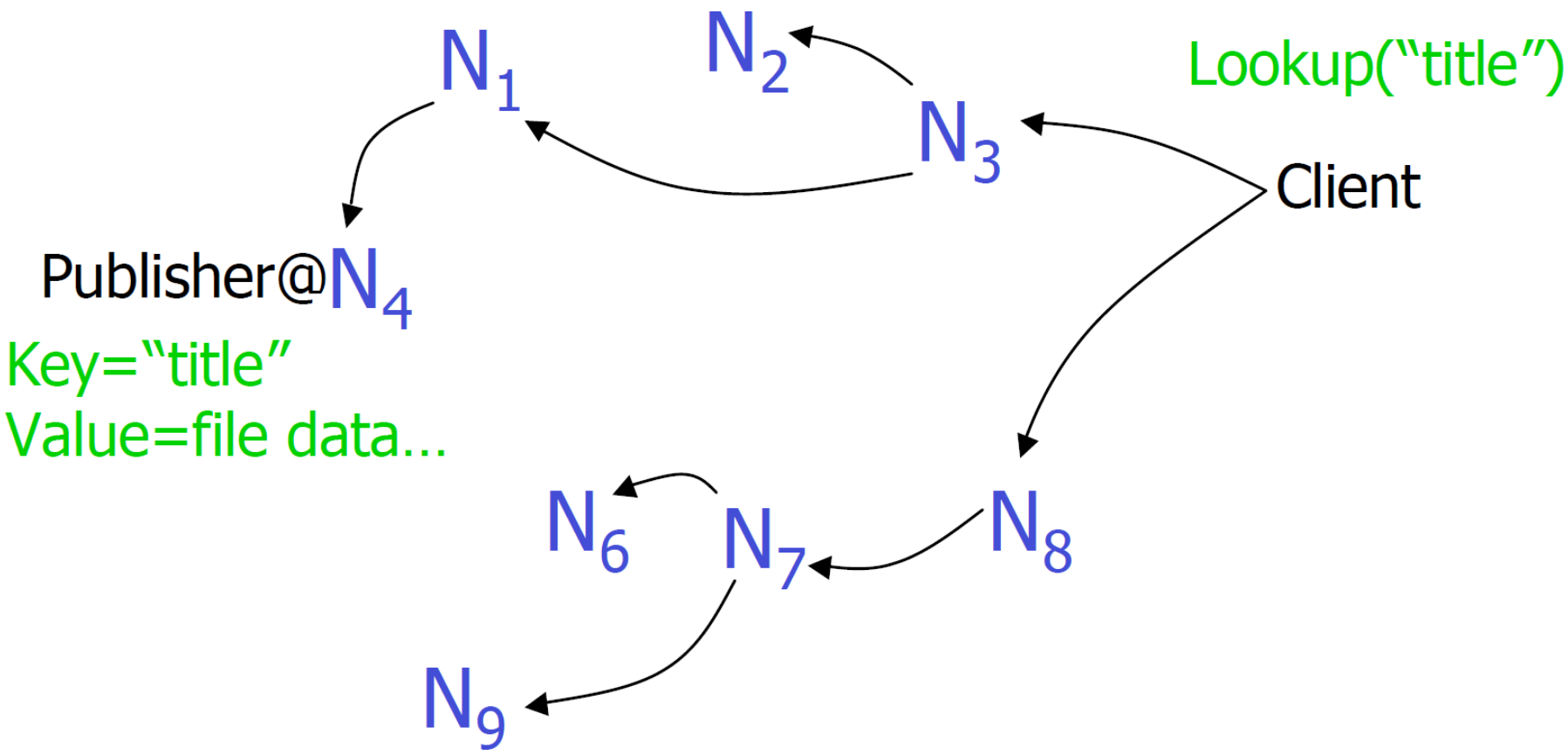
The Lookup Problem – Centralized Lookup (Napster)



Simple, but $O(n)$ state information has to be maintained on a single central node, and the network suffers from a single point of failure.

Routing in P2P Networks

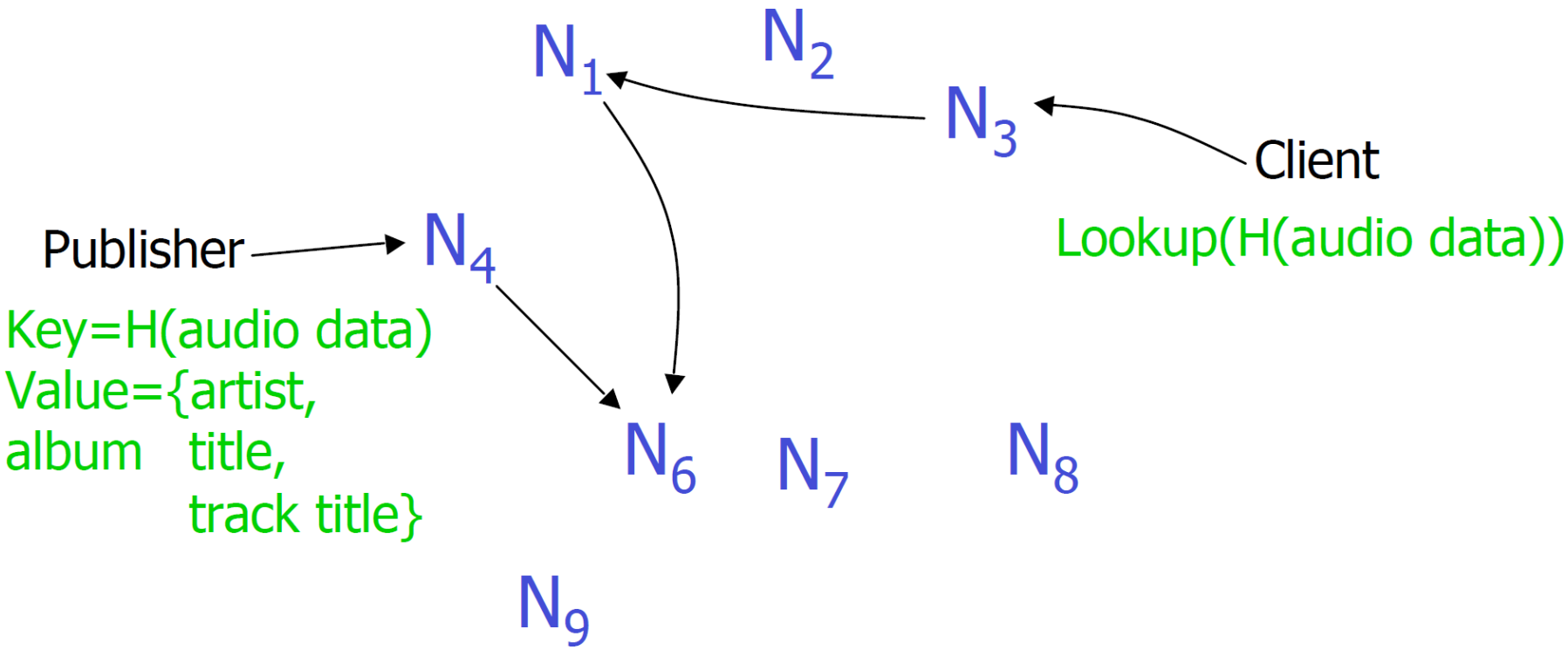
The Lookup Problem – Flooded Queries (Gnutella)



Robust, but in the worst case $O(n)$ messages has to be transmitted per lookup.

Routing in P2P Networks

The Lookup Problem – Routed DHT Queries (Chord, CAN, Pastry, Tapestry, . . .)



Routing in P2P Networks

Evaluation Metrics

The effectiveness/efficiency of a routing scheme can be evaluated by several metrics:

- *Storage*
  - each peer may need to incur some storage space for maintaining metadata (used for searching)
  - storing more metadata $\Rightarrow$ it is more costly to keep these data up-to-date
- *Efficiency*
  - a system is efficient if it can locate the resources quickly
  - metric of efficiency is the response time (can be measured by the average query path length)
- *Usability*
  - reflects the ease of use, and the types of queries that can be supported
  - e.g., depending on the metadata maintained, one system may support complex queries, while another one can perform an exact match only
- *Coverage*
  - refers to whether the search space contains the answers
  - a scheme with a higher coverage is certainly more useful
- *Scalability*
  - important – makes the routing scheme useful in largescale environments
  - a measure of scalability – e.g, the number of messages that need to be routed in order to locate information

Routing in Unstructured P2P Networks

- each peer typically stores its own data objects and selfmaintains a set of links to neighbor nodes
- when a node wants to join the system, it simply contacts an existing node and copies links of that node to form its own links
  - (later maintained independently on the contacted node)
- $\Rightarrow$ no peers have global knowledge of data placement
  - flooding-based techniques have to be used for queries
  - to alleviate the problem of flooding the system with query messages, a *Time-to-Live (TTL)* value is usually attached to each query
  - the challenge is, *how to optimize query processing in the limited number of search steps constrained by TTL*
- several routing strategies have been proposed:
  - *Breadth-First Search (BFS)* – e.g., Gnutella
  - *Depth-First Search (DFS)* – e.g., FreeNet
  - *Heuristic-Based Routing Strategies*

Depth-First Search (DFS)

Algorithm 1 : *FreeNet_Search* (Node x, Key k, TTL t)

```
1:  result = Local_Search(k)
2:  if result = found then
3:      return result to the requester node
4:  else
5:      if t = 0 then
6:          return "not found" to the requester node
7:      else
8:          repeat
9:              pick a neighbor node y in the routing table of x that has the nearest key
                to k and has not been searched before
10:             result = FreeNet_Search(y, k, t − 1)
11:         until result = found or all neighbors have been searched
12:         return result to the requester node
13:     end if
14: end if
```

Figure: *FreeNet's routing strategy:* instead of sending a query to all neighbors, each node selects the most promising neighbor that can answer the query and sends the query to only that node. If the node does not receive a reply within a certain period of time (or the answer cannot be found), the node selects a next promising neighbor.

Heuristic-Based Routing Strategies

Iterative Deepening

- **the idea:**
  - a query is initiated with a sequence of multiple traditional BFS searches by enlarging search radius gradually
  - the search process terminates when either the maximum depth is reached or the results for the query satisfy user's requirements
- **algorithm details:**
  - a system policy P must be provided to specify the sequence of the depths at which the iteration happens
    - $P = D_1, D_2, \ldots, D_n$, where $D_1 < D_2 < \ldots < D_n$
  - under this policy, the source node first sends a query message to the network via BFS search of depth D_1
  - if the result obtained satisfies user's requirements, the query is terminated
  - otherwise, the source node issues another resend query message (with the same query ID) with a BFS depth of D_2
    - the nodes that are less than D_1-hops away from the source node do nothing but just forward the query to their neighbors
    - the further nodes process the query in the same way as in the first iteration
  - similarly for D_3, D_4, etc.
  - if the query is not answered until the depth of D_n, the search process terminates

Heuristic-Based Routing Strategies

Directed BFS and Intelligent Search I.

- **the idea:**
  - in BFS, each node sends the query to all of its neighbors
  - in *Directed BFS*, each node only queries a subset of its neighbors
  - the key point is how to intelligently choose "good" neighbors that would potentially contribute more relevant results for the query
- **details on choosing the neighbors:**
  - each node maintains some statistics of its neighbors: the number of previously answered queries through a neighbor node, the number of results obtained, and the latency in receiving the results
  - based on these statistics, the node can choose the neighbors "intelligently" based on several heuristics, e.g.:
    - choose the one that returned the largest number of results previously
    - choose the one that incurred the least hop-count messages previously
    - choose the one that forwarded the largest number of messages previously
    - choose the one that have shortest message queues
    - etc.

Heuristic-Based Routing Strategies

Directed BFS and Intelligent Search II.

- *Directed BFS*
  - *advantage:* the number of query messages in the network is greatly reduced as compared to standard BFS technique
  - *disadvantage:* the statistics stored about each neighbor are too simple
    - they do not contain information related to the content of queries

- $\Rightarrow$ **Intelligent Search**
  - each peer ranks its neighbors based on their relevances to the query
  - the query is routed only to those neighbors that have high relevances
  - it thus provides a more exact ranking of peers than Directed BFS
    - has good performance in networks that exhibit a high degree of query locality

Heuristic-Based Routing Strategies

Local Indices Search

- **the idea:**
  - each node creates and maintains indices for both its local data and the data on its neighbor nodes that are within a radius of k hops from it
    - if $k = 0$, this method is similar to BFS search (local data index only)
  - the result returned at such a node is the same as the result, which would be returned by processing the query at all the nodes within a radius of k hops from the node
- **details:**
  - the queries are processed based on a global policy P that specifies a list of depths in the search tree where the query is processed
    - just the nodes located at the depth specified in P process the query
    - the other nodes simply forward the query to their neighbors (without processing it)
- *advantage:*
  - reducing the processing cost by limiting the query processing to fewer nodes
- *disadvantages:*
  - higher storage cost (more indices need to be stored at a node)
  - higher update cost for these indices
  - inconsistency/obsolescence of the indices (due to dynamics of the network)

Heuristic-Based Routing Strategies

Random Walk I.

- **the idea:**
  - when a peer issues/receives a query, it randomly selects a neighbor to send or forward the query to
  - this process repeats until the search result is found
    - or TTL expires (if employed) $\Rightarrow$ the result is not found
- **details:**
  - *the main disadvantage:*
    - it suffers from long delays in query processing
  - $\Rightarrow$ *k-walker Random Walk Algorithm*
    - the query initiator (the source node) sends k query messages to its randomly-selected neighbors (instead of just a single one = the original 1-*walker algorithm*)
    - when a node receives a query message (a *walker*), it just follows the basic random walk to randomly select *a single* neighbor to forward the query to
    - the number of messages (visited nodes) increases *linearly* as compared to the 1-walker algorithm

Heuristic-Based Routing Strategies

Random Walk II.

- **details cont'd.:**
  - $\Rightarrow$ *Random Breadth First Search (RBFS)*
    - similar to the *k*-walker Random Walk
    - the query initiator first randomly selects a subset of its neighbors to send the query to
    - each of these neighbors then randomly selects *a subset* of its neighbors, where the query is forwarded
    - etc.
    - the number of messages (visited nodes) increases *exponentially* as compared to the 1-walker algorithm

Heuristic-Based Routing Strategies

Adaptive Probabilistic Search (APS)

- **the idea:**
  - a search method that combines techniques of both k-walker random search and probabilistic search
  - the main difference between APS and random walkers:
    - random walkers send the query to random neighbors while APS sends the query to neighbor nodes based on some probabilities
    - $\Rightarrow$ each peer contains a probability for each neighbor with respect to each object (determined from past results)
- **details:**
  - two approaches to update the probabilities:
    - *Optimistic approach* – the system proactively increases the probabilities for selected (= queried) neighbors along the search path and decreases their probabilities only if the walker passing through them terminates with a failure
    - *Pessimistic approach* – the system proactively decreases the probabilities for selected (= queried) neighbors along the search path and increases their probabilities when the walker passing through them terminates with a success
  - *swapping-APS* – each peer swaps between optimistic and pessimistic method
    - based on an observation of the ratio of successful walkers for each object
  - *weighted-APS* – takes into account the location of objects

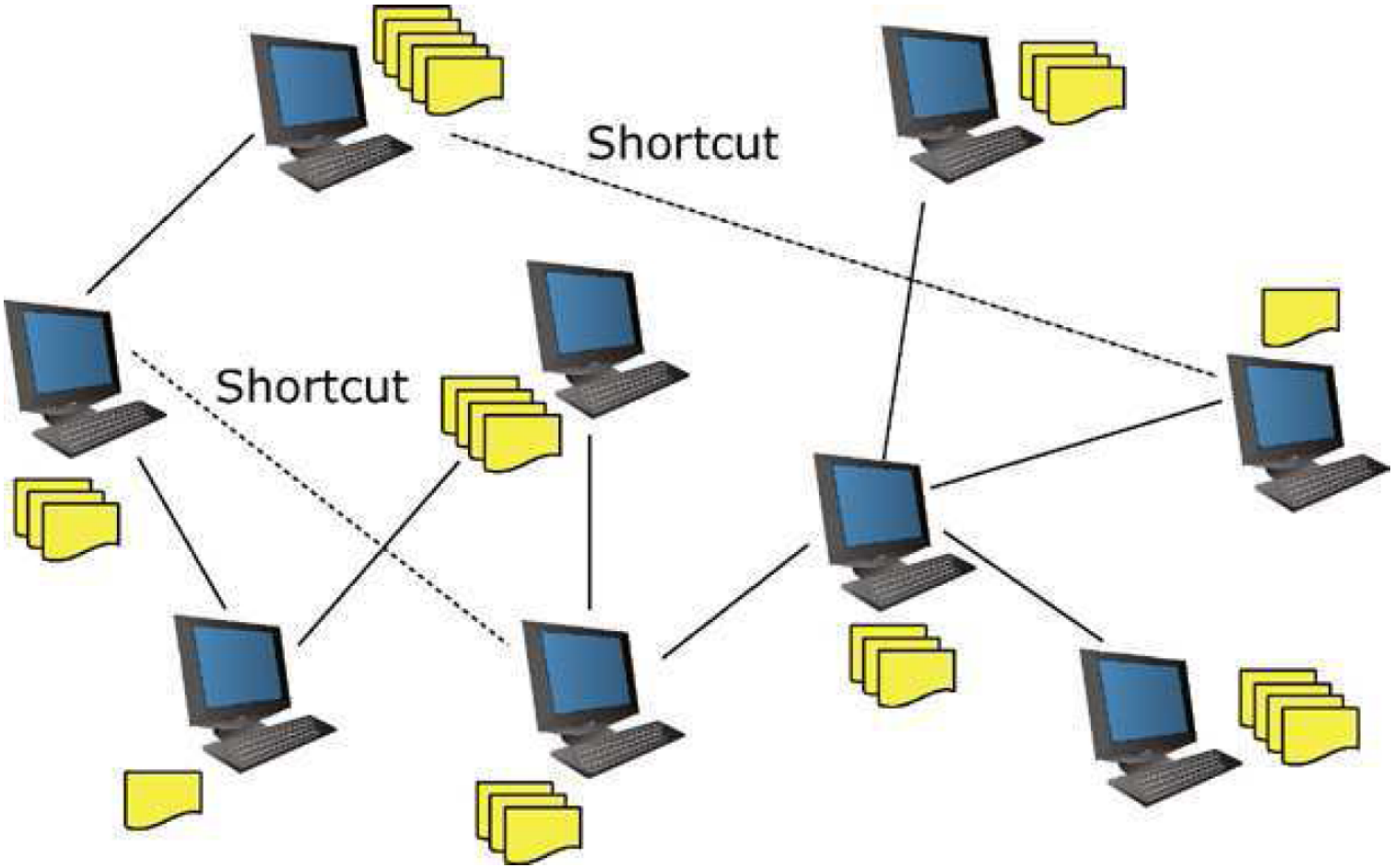
Heuristic-Based Routing Strategies

Interest-Based Shortcuts I.

- **the idea:**
  - each peer adds additional links on top of an existing searching network to improve the search performance
    - these links (called *interest-based short-cuts*) connect two peers having a similar interest
- **details:**
  - when a peer issues a query, it first employs interest-based shortcuts to forward and process the query
    - if the result is found, the search terminates
    - otherwise, the normal query processing algorithm is used
  - *shortcut construction:*
    - when a peer joins the system, it has no shortcuts
    - after each successfully processed query, the query initiator adds shortcuts to peers providing the answers for that query
    - each peer stores only a limited number of shortcuts that have the highest utility (due to space constraints)

Heuristic-Based Routing Strategies

Interest-Based Shortcuts II.



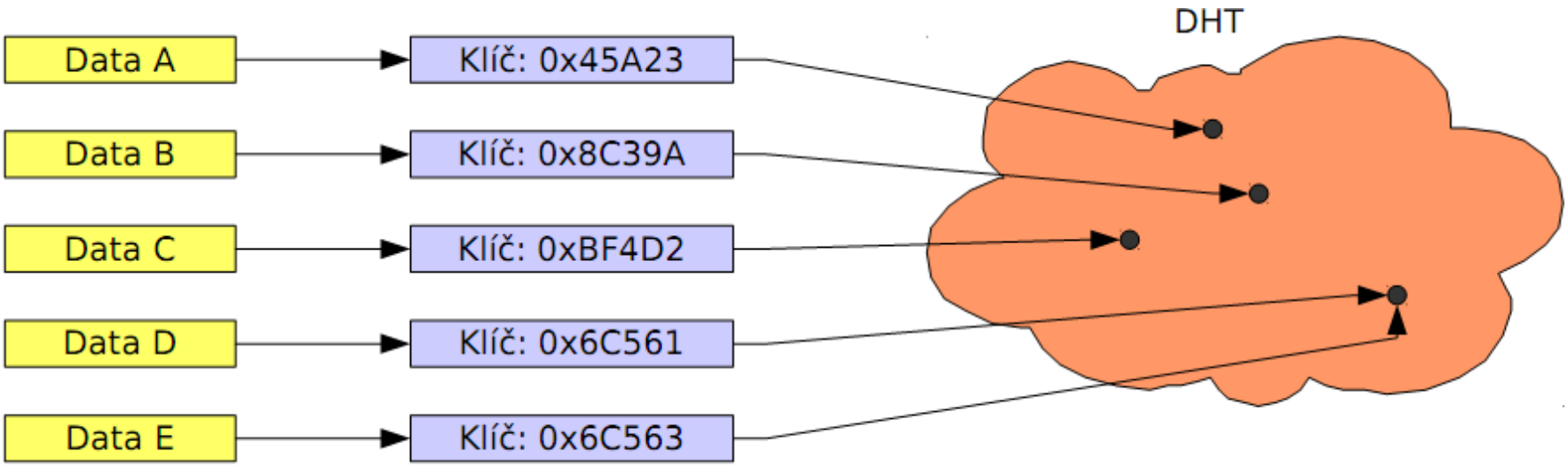
Routing in Structured P2P Networks

- the unstructured P2P networks suffer from the problem of low searching efficiency
- unlike in the unstructured P2P systems, participant nodes in a structured P2P system are required to organize into some fixed topologies
  - such as a ring (Chord), a multidimensional grid (CAN), a mesh (Pastry and Tapestry), or a multiple list (Skip Graph)
  - $\Rightarrow$ when a node joins the system, it has to follow some strict procedures to set up its position
- can be guaranteed, that if a result of a query exists in the system, it will be found
  - moreover, in an efficient way – most systems can provide an answer for a query within $O(log\ N)$ steps/messages (N = number of nodes)
- *disadvantage:*
  - the need for a network topology incurs high maintenance cost (changes in routing tables)
- based on the overlay network structure, structured P2P systems can be classified into the following categories:
  - *Distributed Hash Table (DHT) based systems* – e.g., Chord, CAN, Tapestry and Pastry, Viceroy and Crescendo, etc.
  - *Skip List based systems* – e.g., Skip Graph, SkipNet, etc.
  - *Tree based systems* – e.g., P-Grid, P-Tree, BATON, etc.

Distributed Hash Table (DHT) based P2P systems

Distributed Hash Table

- every node in the P2P network manages its part of global hash table
- storage/retrieval of an item s means quering the node, which manages the part, where the $hash(s)$ belongs to



Distributed Hash Table (DHT) based P2P systems

Chord I.

- one of the most widely known routing mechanism in structured P2P networks
- **the idea:**
  - uses a one-way consistent hash function to map each node and data item to an m-bit identifier in a single-dimensional identifier space
    - the hash function uses the node's IP address to generate an identifier for a node, and
    - the data item (or the key of the data item) to generate an identifier for the data
    - the identifier space must be chosen large enough (the probability of assigning the same identifier to different nodes should be negligible)
- **details:**
  - the identifier space is a circle of numbers from 0 to $2^m - 1$
  - the system assigns a key k to the first node n whose identifier is equal to or follows the identifier k in the circle space
    - i.e., the key k is assigned to the first node clockwise from k

Distributed Hash Table (DHT) based P2P systems

Chord II.

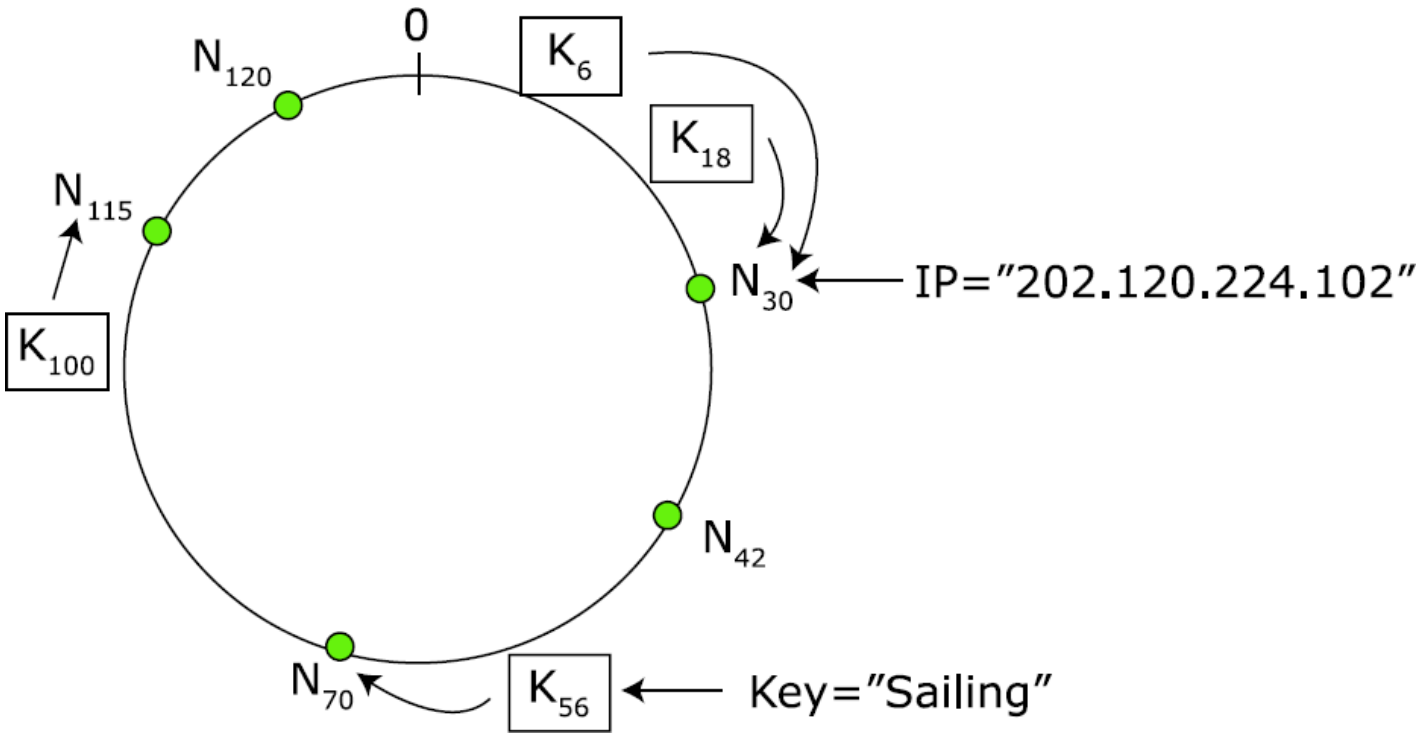


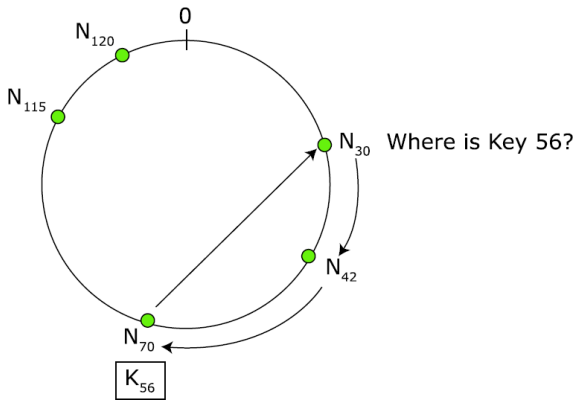
Figure: An identifier circle based on consistent hashing – keys K_6 and K_{18} are assigned to the same node identifier N_{30} (obtained by hashing the IP address "202.120.224.102"). The key K_{56} (obtained by hashing the word "Sailing") is assigned to the node identifier N_{70}; the key K_{100} is assigned to the node identifier N_{115}; the nodes N_{42} and N_{120} store no data items.

Distributed Hash Table (DHT) based P2P systems

Chord III. – Simple lookup algorithm

Simple lookup algorithm:

- each node only needs to know its *immediate successor node*
- when a node receives a query request:
  - first, it checks its local storage to see if it holds the queried data item
    - if yes, the result is returned to query sender
  - if no, it forwards the query to its immediate successor node
  - the lookup terminates, when
    - the result is found
    - the identifier of a node's immediate successor exceeds the identifier of the queried data item $\Rightarrow$ the result cannot be found
  - the complexity is $O(N)$ (N = the number of nodes in the system)



Distributed Hash Table (DHT) based P2P systems

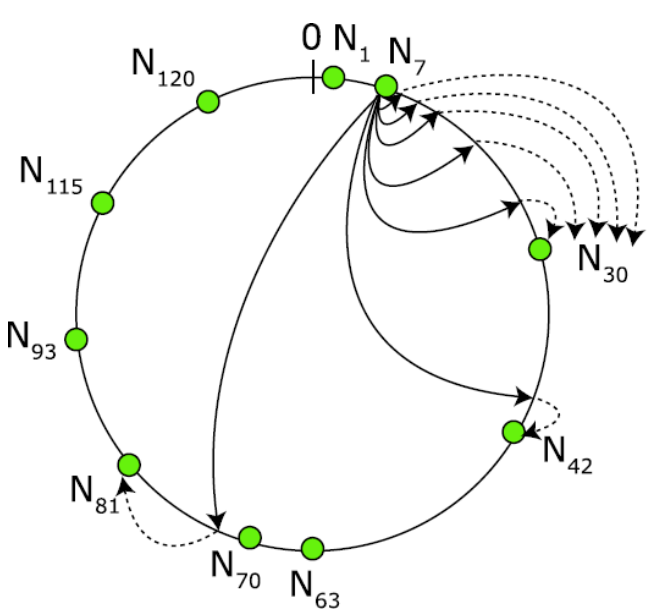
Chord III. – Scalable lookup algorithm I.

Scalable lookup algorithm:

- instead of maintaining only a single immediate successor node, each node maintains a *finger table consisting of m successor nodes*
- when a node n receives a query request:
  - if the node does not hold the queried data, it searches its finger table for a node n' with the highest node identifier that satisfies the condition $n.id < n'.id < k$
    - if such a node exists, the node n asks n' to find the key k
    - otherwise, the node n asks its immediate successor to find k
  - the lookup terminates, when
    - the result is found
    - the identifier of a node's immediate successor exceeds the identifier of the queried data item $\Rightarrow$ the result cannot be found
  - the complexity is $O(log\ N)$ (N = the number of nodes in the system)

Distributed Hash Table (DHT) based P2P systems

Chord III. – Scalable lookup algorithm II.



Finger Table

N_{7+1}	N_{30}
N_{7+2}	N_{30}
N_{7+4}	N_{30}
N_{7+8}	N_{30}
N_{7+16}	N_{30}
N_{7+32}	N_{42}
N_{7+64}	N_{81}

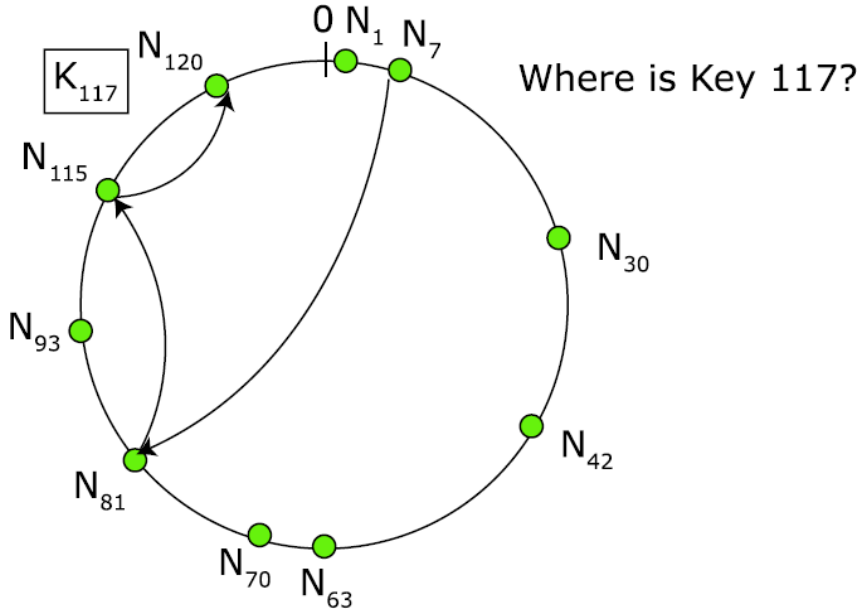


Figure: An example of finger table entries (left) and an example of a routing path for key K_{117} starting at node N_7 (right).

Distributed Hash Table (DHT) based P2P systems

Chord IV.

System construction:

- when a new node joins the system, it needs to:
  1. find its position in the Chord ring and obtain data it should be responsible for (based on keys)
  2. initialize its finger table
  3. update finger tables of other nodes to reflect the presence of it
- when an existing node leaves the system, it does not need to do anything

Distributed Hash Table (DHT) based P2P systems

Content Addressable Network (CAN) I.

- **the idea:**
  - a routing system built on a virtual d-dimensional Cartesian coordinate space
    - the system partitions the storage space into different zones, each of which is assigned to a node
    - such a node stores all data items belonging to its zone
    - the system uses a uniform hash function to map the data key value to a point p in the coordinate space (thus obtaining a d-tuple)
- **details:**
  - inserting a data item:
    1. the data key value is mapped into a point p in the coordinate space
    2. the node n, whose zone covers p, is found and contacted to store the new data item
  - processing a query is similar
    - if the result exists, it should be stored on the node covering the particular zone
  - each node needs to maintain information about its neighbor nodes
    - i.e., the nodes covering adjacent zones
  - the routing is based on a simple greedy forwarding algorithm
    - in every step, a node having closer coordinates to the destination zone is chosen

Distributed Hash Table (DHT) based P2P systems

Content Addressable Network (CAN) II.

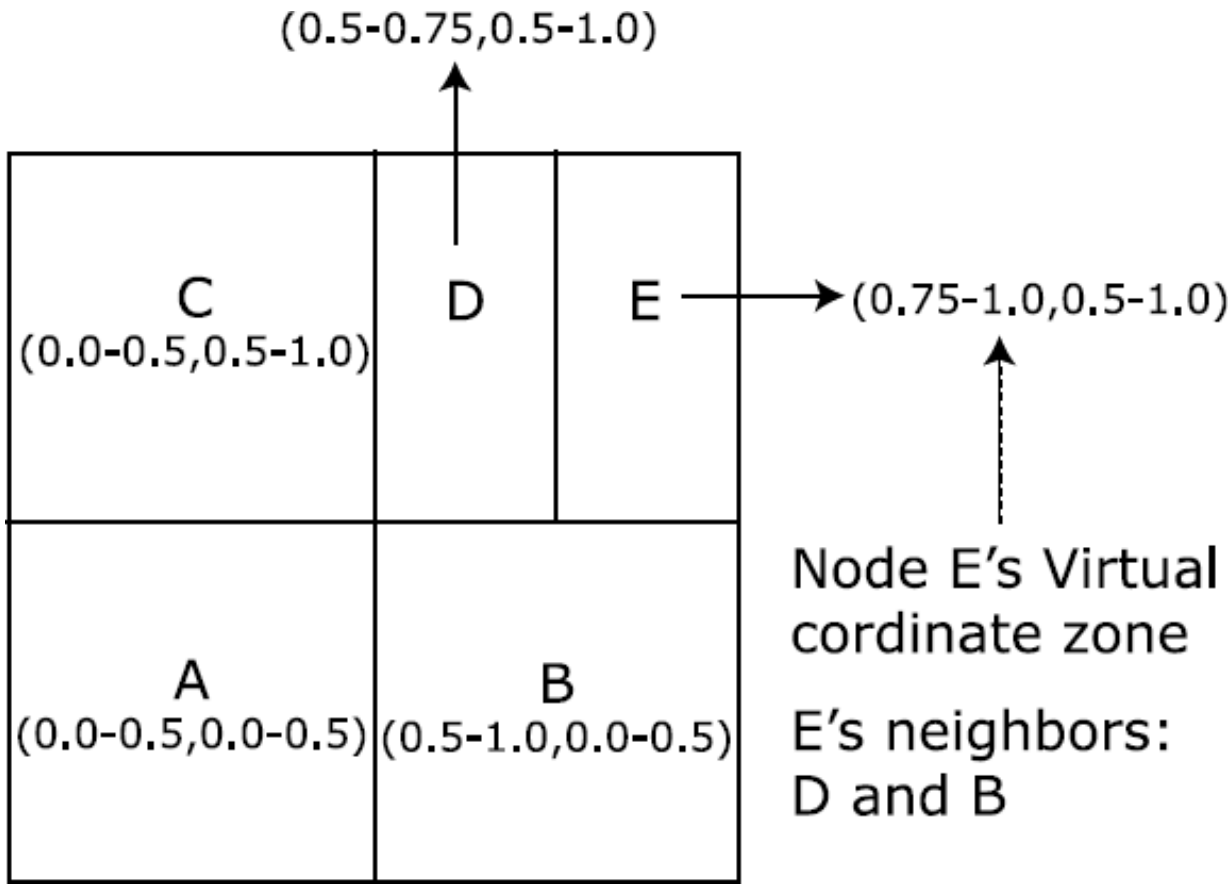


Figure: A CAN system using two dimensional space with 5 nodes.

Distributed Hash Table (DHT) based P2P systems

Content Addressable Network (CAN) III.

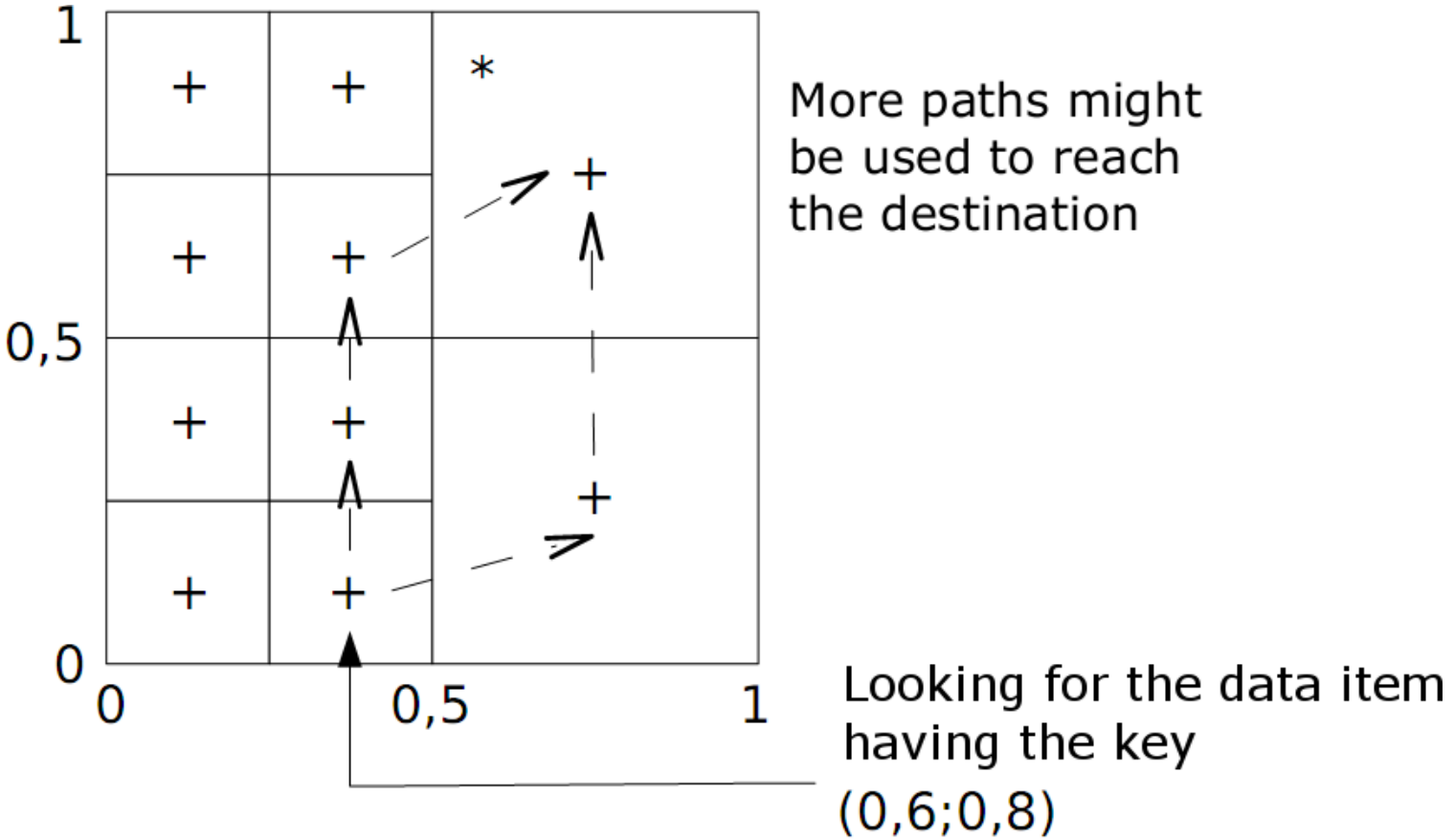


Figure: An example of a data item lookup in a CAN system.

Distributed Hash Table (DHT) based P2P systems

Content Addressable Network (CAN) IV.

System construction:

- when a new node joins the system, it needs to:
  1. find an arbitrary node, which is already connected to the network
  2. identify a zone, which might be divided, and ask its owner/maintainer node to split the zone into two parts
     - the original node keeps maintaining one part, the new node starts to maintain the second one
  3. construct its own routing table and update the routing tables of its neighbors
- when an existing node leaves the system, it has to ask its neighbor to merge the zones into a single one

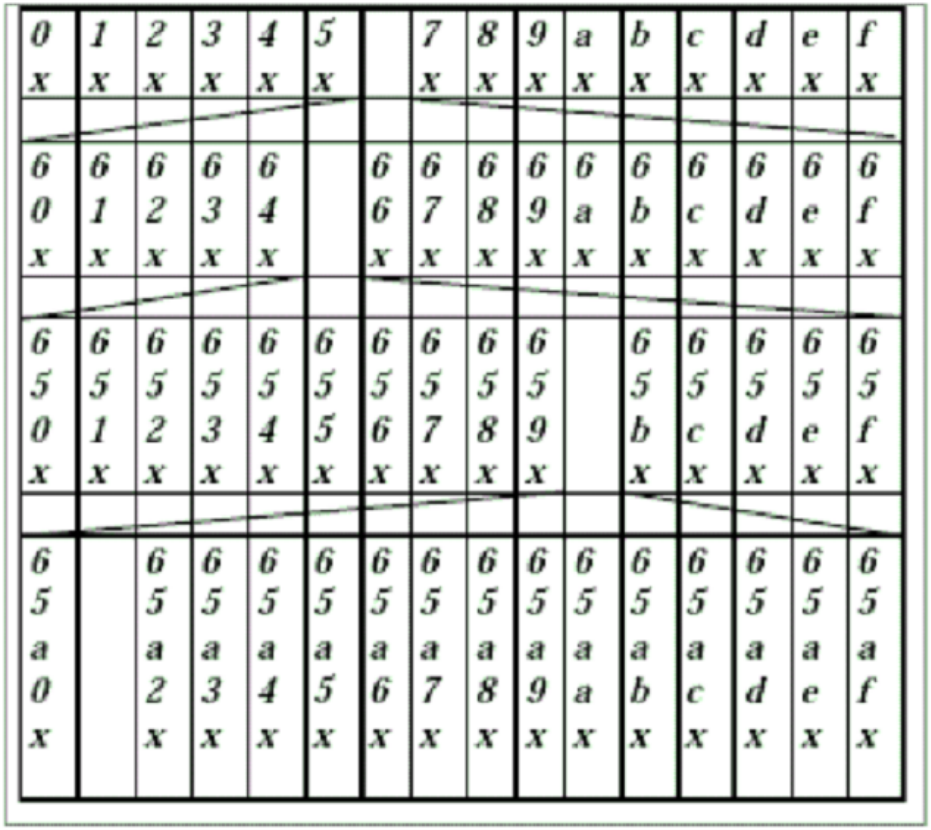
Distributed Hash Table (DHT) based P2P systems

Pastry I.

- **the idea:**
  - a routing system based on *PRR trees*
    - PRR = Plaxton, Rajaraman, and Richa (1997)
  - a node identifier is an m-bit number broken up into a sequence of digits having the base 2^b
    - e.g., a 128-bit identifier is broken up into 32 4-bit digits ($b = 4$, base $= 2^4 \Rightarrow$ hexadecimal sequence of digits)
    - b . . . configuration parameter
  - a data item is stored on a node having the identifier, which shares the *longest prefix with the data identifier*
  - in every routing step, a neighbor node having a longer prefix in common with the destination node (longer by 1 digit, i.e., b bits) is chosen
    - the routing complexity is $O(log_{2^b} N)$
- **details:**
  - each peer has a *routing table* to route messages
    - organized in a fixed number of levels ($= \lceil log_{2^b}(N) \rceil$) and within each level a fixed number of entries ($= 2^b - 1$)
    - row ID = the length of prefix in common with the destination node
    - column ID = next possible step

Distributed Hash Table (DHT) based P2P systems

Pastry II.



0	1	2	3	4	5		7	8	9	a	b	c	d	e	f
x	x	x	x	x	x		x	x	x	x	x	x	x	x	x
6	6	6	6	6		6	6	6	6	6	6	6	6	6	6
0	1	2	3	4		6	7	8	9	a	b	c	d	e	f
x	x	x	x	x		x	x	x	x	x	x	x	x	x	x
6	6	6	6	6	6	6	6	6	6		6	6	6	6	6
5	5	5	5	5	5	5	5	5	5		5	5	5	5	5
0	1	2	3	4	5	6	7	8	9		b	c	d	e	f
x	x	x	x	x	x	x	x	x	x		x	x	x	x	x
6		6	6	6	6	6	6	6	6	6	6	6	6	6	6
5		5	5	5	5	5	5	5	5	5	5	5	5	5	5
a		a	a	a	a	a	a	a	a	a	a	a	a	a	a
0		2	3	4	5	6	7	8	9	a	b	c	d	e	f
x		x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figure: A routing table for the node **65a1**. The x represents a routing information (next step) for the node having the particular prefix. (White spaces correspond to prefixes identical with the actual node.)

Distributed Hash Table (DHT) based P2P systems

Pastry III.

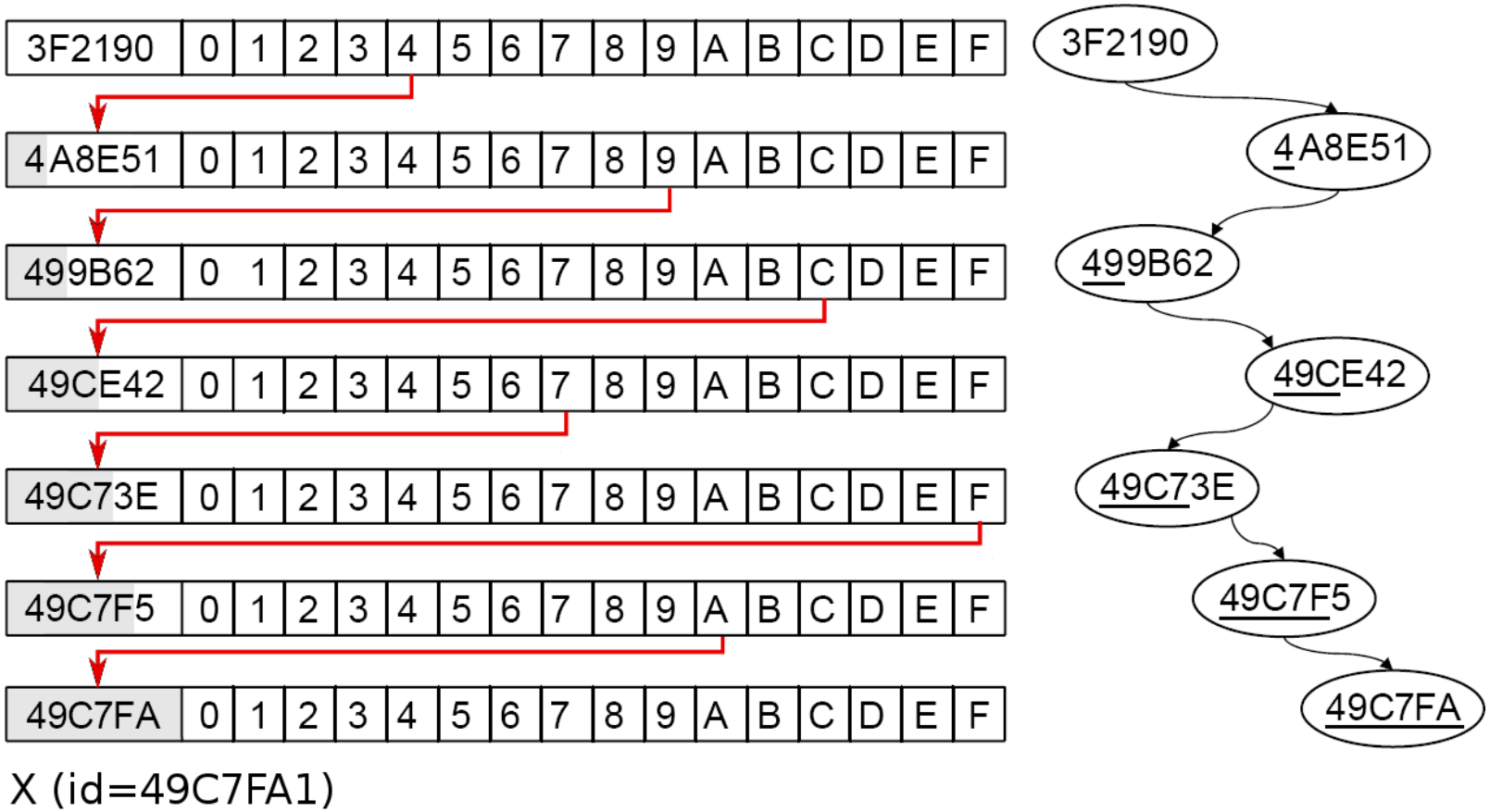


Figure: An example of inserting/searching an object X with an identifier $49C7FA1$ into a Pastry network (starting at the node $3F2190$).

Distributed Hash Table (DHT) based P2P systems

Pastry IV.

- **details cont'd:**
  - besides the routing table, each node has a *leaf set* as well
    - contains a list of pointers to the nodes, which are numerically closest to the particular node
    - serves as a fall back when a node with longer prefix is not found in the routing table
    - it's size is typically 2^b – a half of the table contains keys lower than the particular node's key, the other half contains keys greater than the particular node's key
- **routing in Pastry:**
  - if the searched node is in the leaf set of the particular node, the routing is performed based on it
  - otherwise, the routing table is consulted
    - the message is forwarded to a node that shares a most common prefix with the key
  - if the routing table is empty or the referenced node cannot be reached, the message is forwarded to:
    - a neighbor having the same common prefix length with the key as the node, or
    - a node whose identifier is numerically closer to the key than the node's id.
    - (very rare case)

Distributed Hash Table (DHT) based P2P systems

Pastry V.

System construction:

- when a new node (having an identifier X) joins the network:
  1. it has to contact an arbitrary node A (being already connected), and send a message *join(X)* to it
  2. the node A routes the message *join(X)* to the node Z, which is the closest one to the key X
  3. the node X receives a leaf set from the node Z and fills in its routing table (the table's i-th row is received from the i-th node on the path from A to Z)
  4. the node X informs the nodes, which should insert it into their routing tables
- when an existing node leaves the network:
  - it has to pass the data it has managed to a neighbor
  - the routing tables become automatically updated soon
    - the node becomes replaced with a node from its leaf set (one of its neighbors)

Distributed Hash Table (DHT) based P2P systems

Tapestry

Tapestry:

- another peer-to-peer overlay routing infrastructure based on PRR Trees, which is very similar to Pastry
- the main difference between Pastry and Tapestry:
  - in Pastry, each routing hop extends the matching *prefix*
  - in Tapestry, each routing hop extends the matching *suffix*
  - (another slight differences also exist)

Distributed Hash Table (DHT) based P2P systems

Comparison

	CAN	Chord	Pastry
Routing performance	$O(d * N^{1/d})$	$O(\log N)$	$O(\log_B N)$
Routing state	2 d	log N	$B * \log_B N + B$
Peers join/leave	2 d	$(\log N)^2$	$\log_B N$

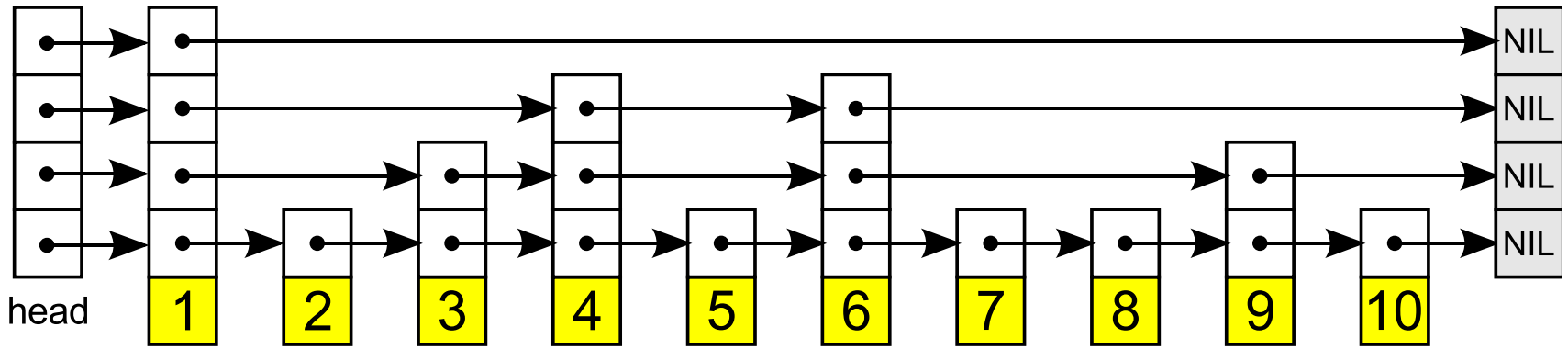
$B = 2^b$

Figure: The comparison of presented DHT-based routing mechanisms for structured P2P networks (the lookup performance view, the storage view, and the re-management during a node's join/leave view).

Skip List based P2P systems

Skip List structure I.

- a **skip list** is a data structure for storing a sorted list of items using a hierarchy of linked lists
  - the lists connect increasingly sparse subsequences of the items
- the lists are built in layers:
  - the bottom layer (level 0) is an ordinary ordered linked list
  - each higher layer acts as an "express lane" for the lists below, where an element in layer i appears in layer $i + 1$ with some fixed probability p
    - usually, $p = 1/2$ or $p = 1/4$



Skip List based P2P systems

Skip List structure II.

A search for a target element:

- begins at the head element in the top list and proceeds horizontally until the current element is greater than or equal to the target
  - if the current element is equal to the target, the target has been found
  - if the current element is greater than the target, the procedure is repeated after returning to the previous element and dropping down vertically to the next lower list
- the *expected cost of a search* is $(log_{1/p}\, n)/p$
  - since p is a constant $\Rightarrow O(log\, n)$

Skip List based P2P systems

Skip List structure III.

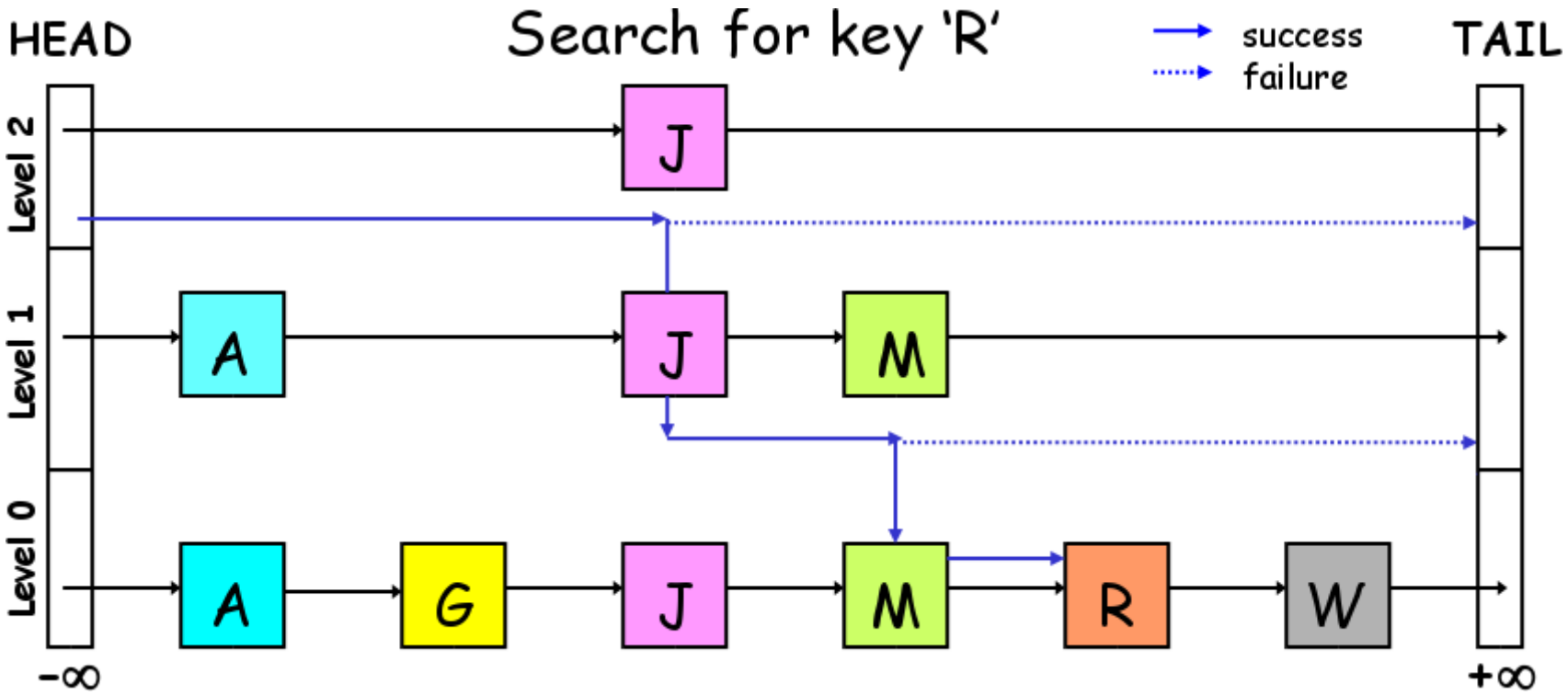


Figure: The searching process in a Skip List structure.

Skip List based P2P systems

Skip Graph I.

the idea:

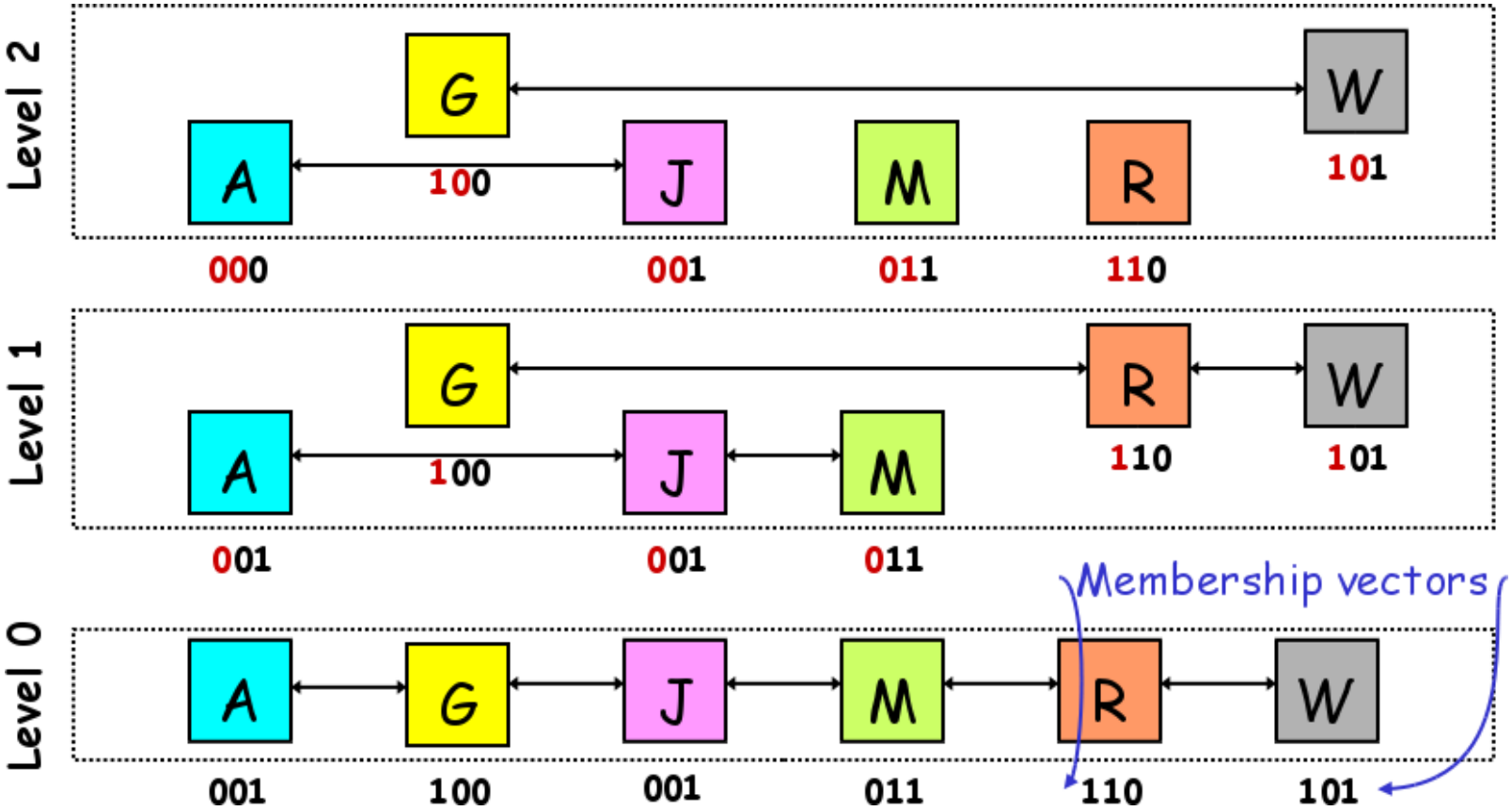
- a routing system based on *Skip Lists*
  - pure Skip Lists are not suitable, since the top-level nodes may become overloaded
- unlike pure Skip List, which has only one list at each level, a Skip Graph has *many lists at each level*
  - each node participates in a list at each level
  - the system controls the lists, which a node belongs to, by a random *membership vector* (created when the node joins the system)
  - the number of levels is $O(log\ N)$

lookup details:

- once a node issues a query:
  - the search process always starts at the highest level of that node
  - at each step, if there is a neighbor node at the same level that keeps a closer value to the search key, the node forwards the query to that neighbor
  - otherwise, the node continues the search process at a lower lever
  - the destination node containing the result is found when the search process reaches the bottom level
- the query processing complexity is $O(log\ N)$

Skip List based P2P systems

Skip Graph II.

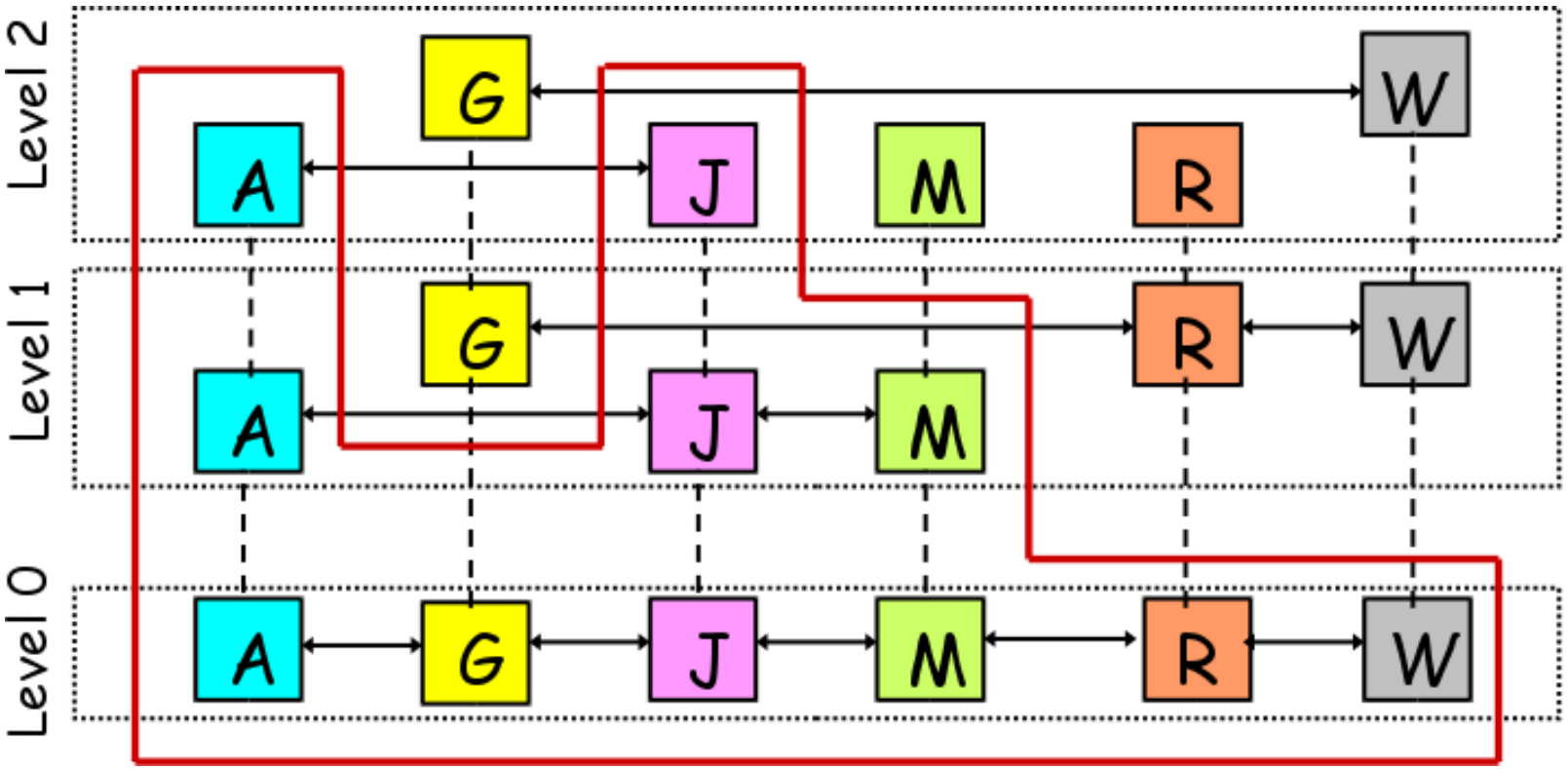


The **Membership vector** only defines, which lists the particular element belongs to (*the lists are sorted by a data key*).

Skip List based P2P systems

Skip Graph III.

Restricting to the lists containing the starting element of the search, we get a *skip list* (the **pure skip list searching method can be used** then):



Skip List based P2P systems

Skip Graph IV.

System construction:

- when a new node (having an identifier X) joins the network:
  - based on its membership vector $m(X)$, X joins the lists of nodes whose membership vector shares the same prefix with $m(X)$ at different lengths
  - in particular:
    - X first joins the list at level 0 (to the nodes containing keys closest to the X's key)
    - for every level $i \geq 1$, X links to the closest node Y having the same i-length prefix with the node X

Skip List based P2P systems

Skip Graph V.

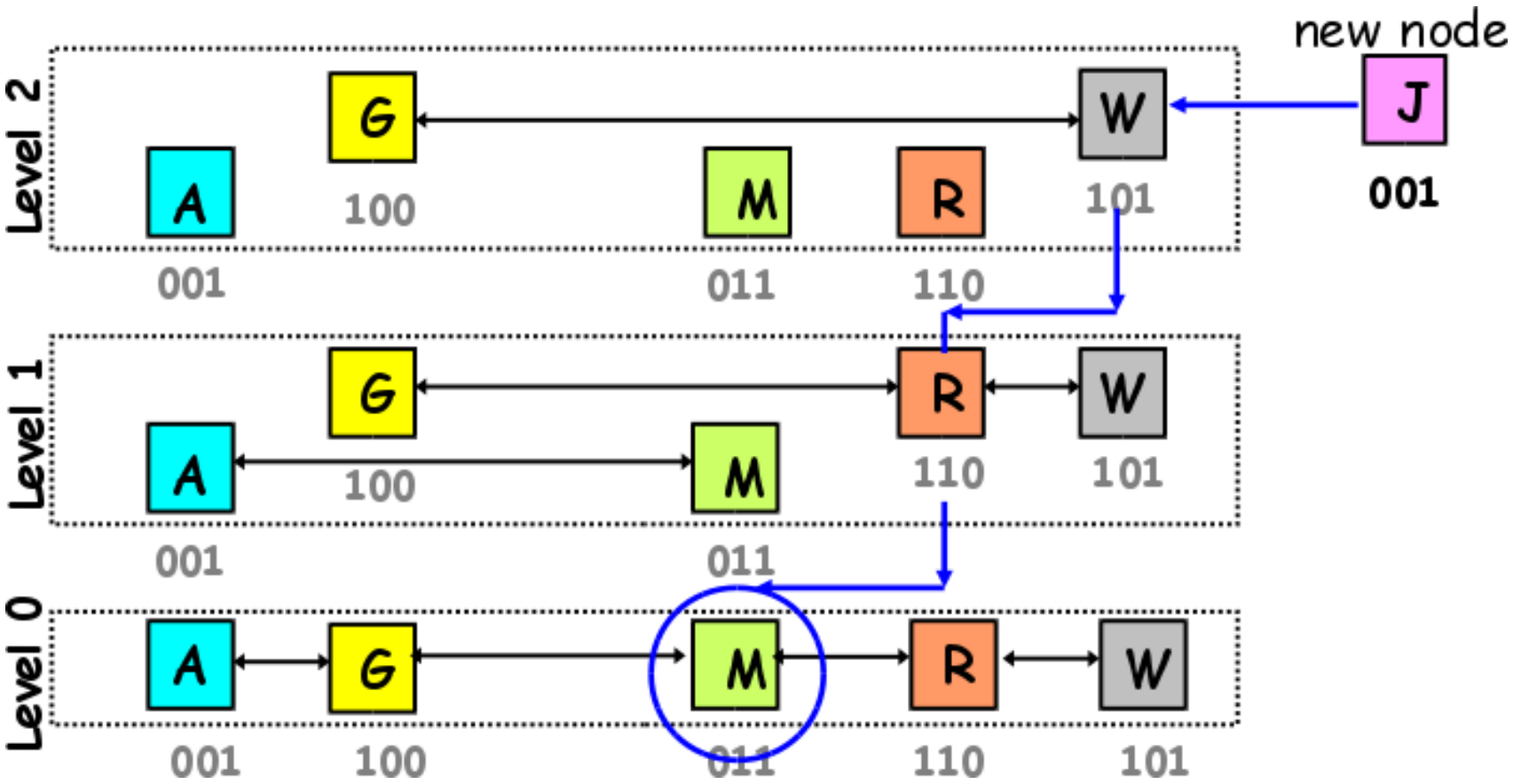


Figure: *Step 1:* Starting at an arbitrary node, find a nearest **(data) key** at level 0.

Skip List based P2P systems

Skip Graph VI.

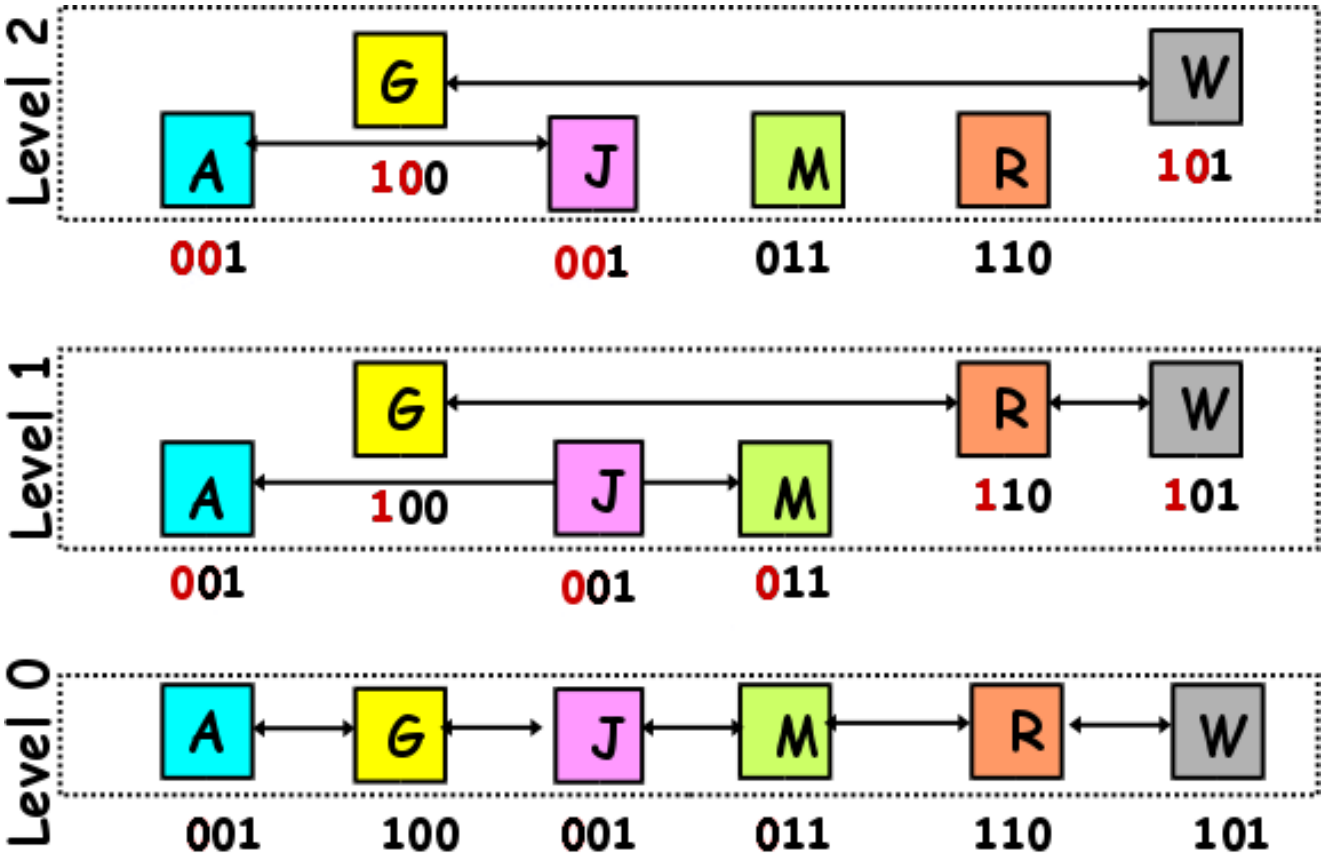


Figure: *Step 2:* At each level i, connect to the list with a matching prefix of the membership vector of length i.

Skip List based P2P systems

SkipNet I.

the idea:

- a routing system very similar to Skip Graph
- instead of Skip Lists, the SkipNet organizes nodes into *rings*
  - similarly to the Skip Graph, organized into levels as well
  - the nodes are *sorted on each level based on a data key*
  - on a particular level, every node has a pointer to its neighbors stored in its routing table
    - the pointers on the level h point to the nodes that are roughly 2^h nodes to the left and right of the given node
    - all the nodes are connected by the *root ring* formed at level 0
- the routing/lookup mechanism and system construction are very similar to the Skip Graph's ones

Skip List based P2P systems

SkipNet II.

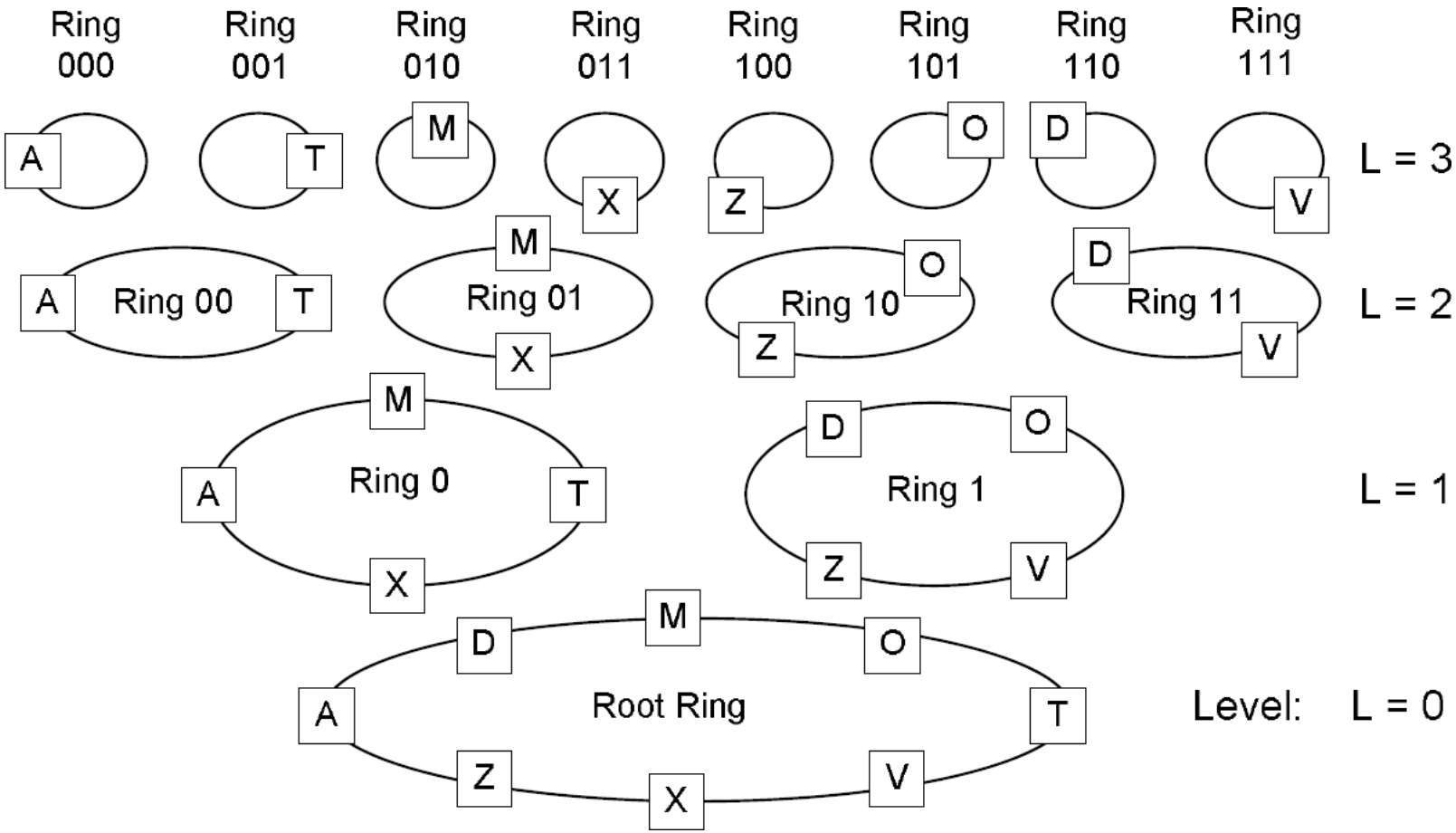
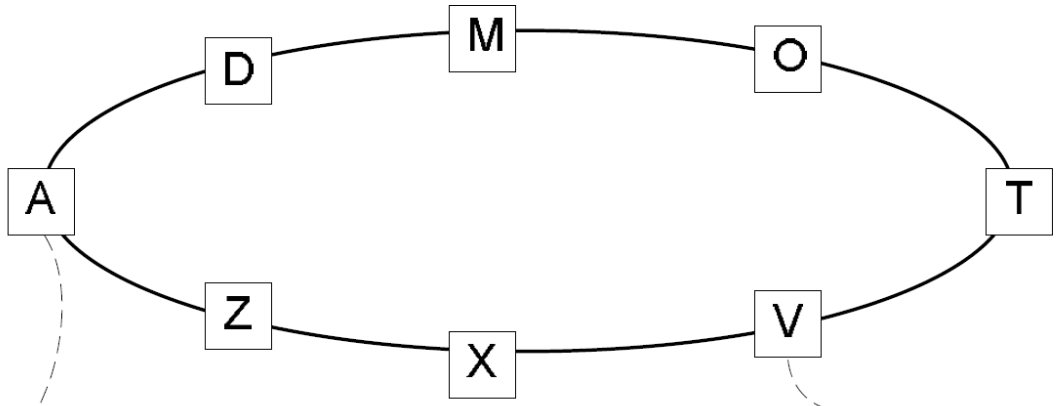


Figure: The full SkipNet routing infrastructure for an 8 node system, including the ring labels.

Skip List based P2P systems

SkipNet III.



Level		
2	T	T
1	M	X
0	D	Z

Level		
2	D	D
1	Z	O
0	X	T

Figure: The routing tables for nodes A and V.

Skip List based P2P systems

SkipNet IV.

A routing example: Routing from *A* to *V*

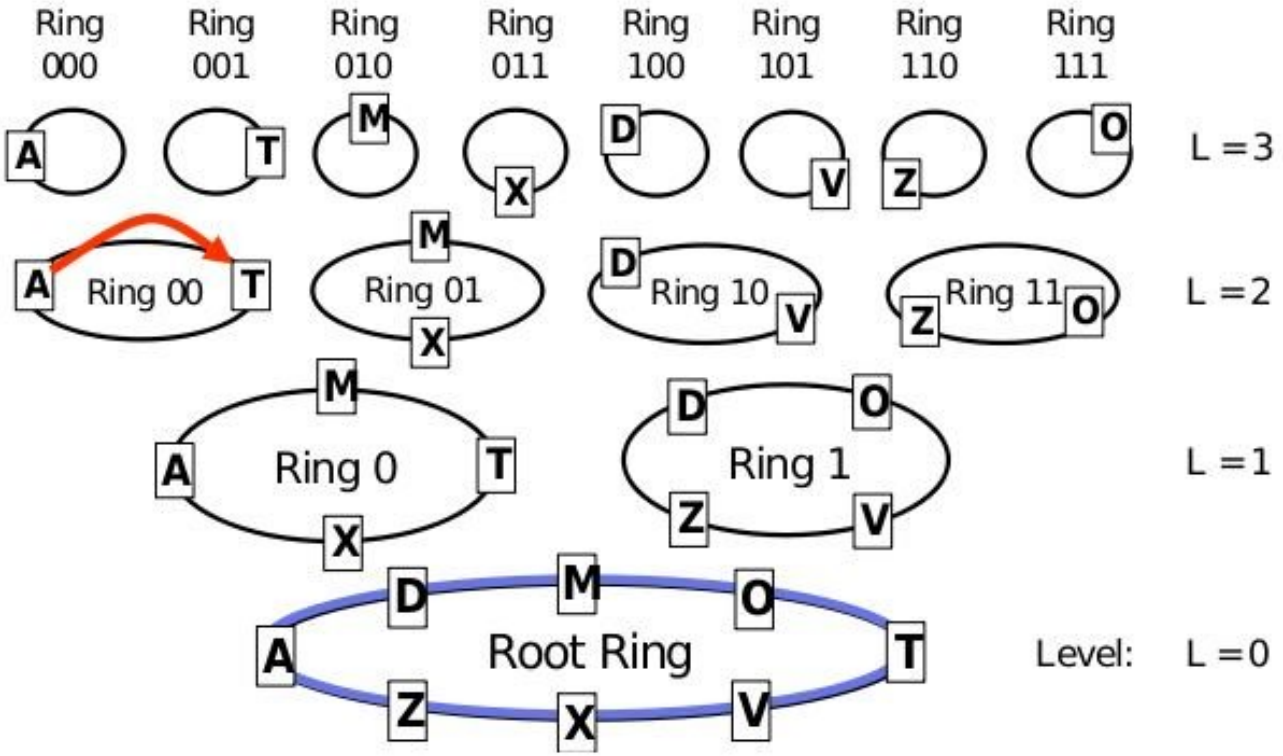


Figure: At first, the message is forwarded to a neighbor closer to the destination.

Skip List based P2P systems

SkipNet V.

A routing example: Routing from *A* to *V*

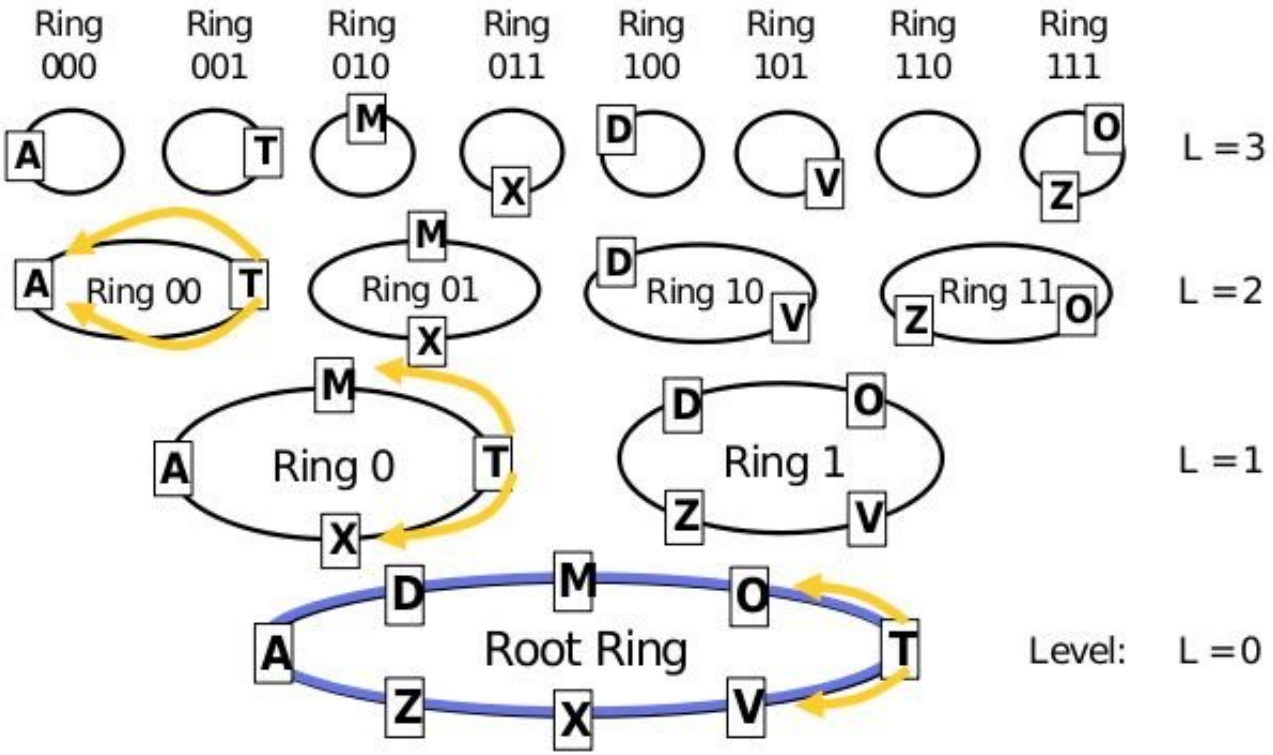


Figure: Node *T*'s routing table.

Skip List based P2P systems

SkipNet VI.

A routing example: Routing from *A* to *V*

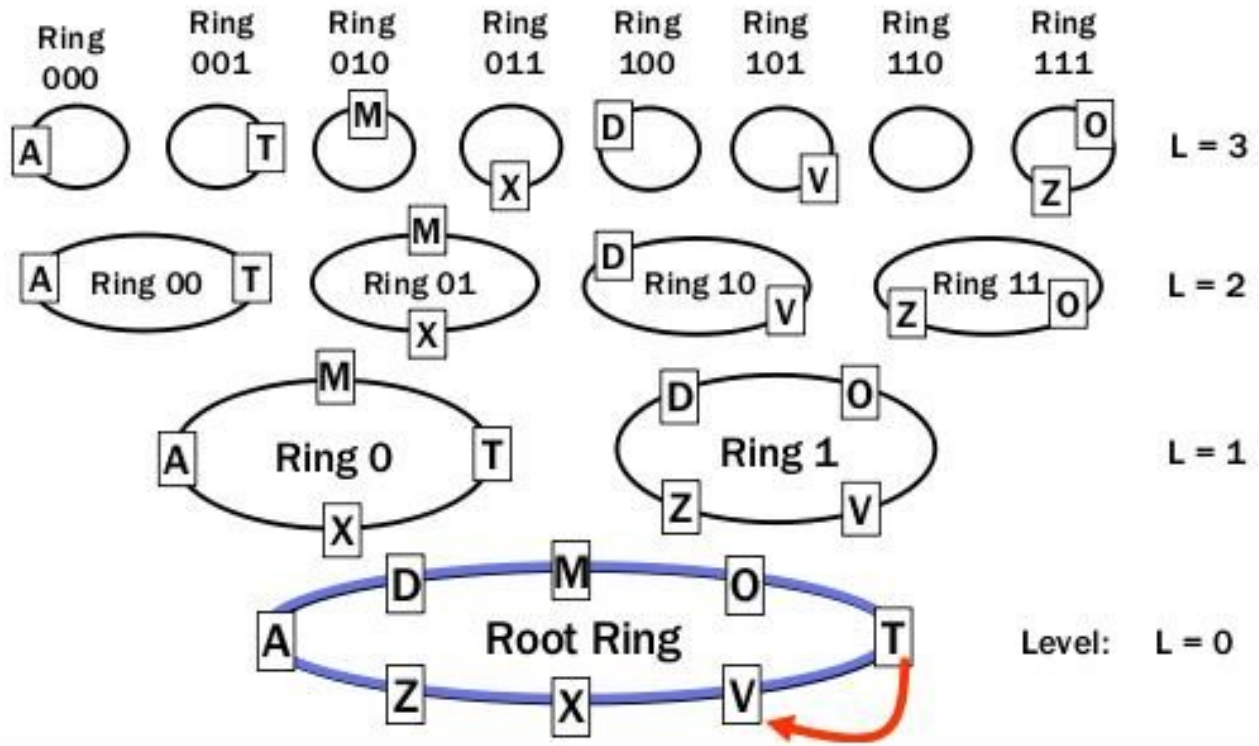


Figure: Since there is a direct access to the node *V* at level 0, the lookup terminates.

Tree based systems

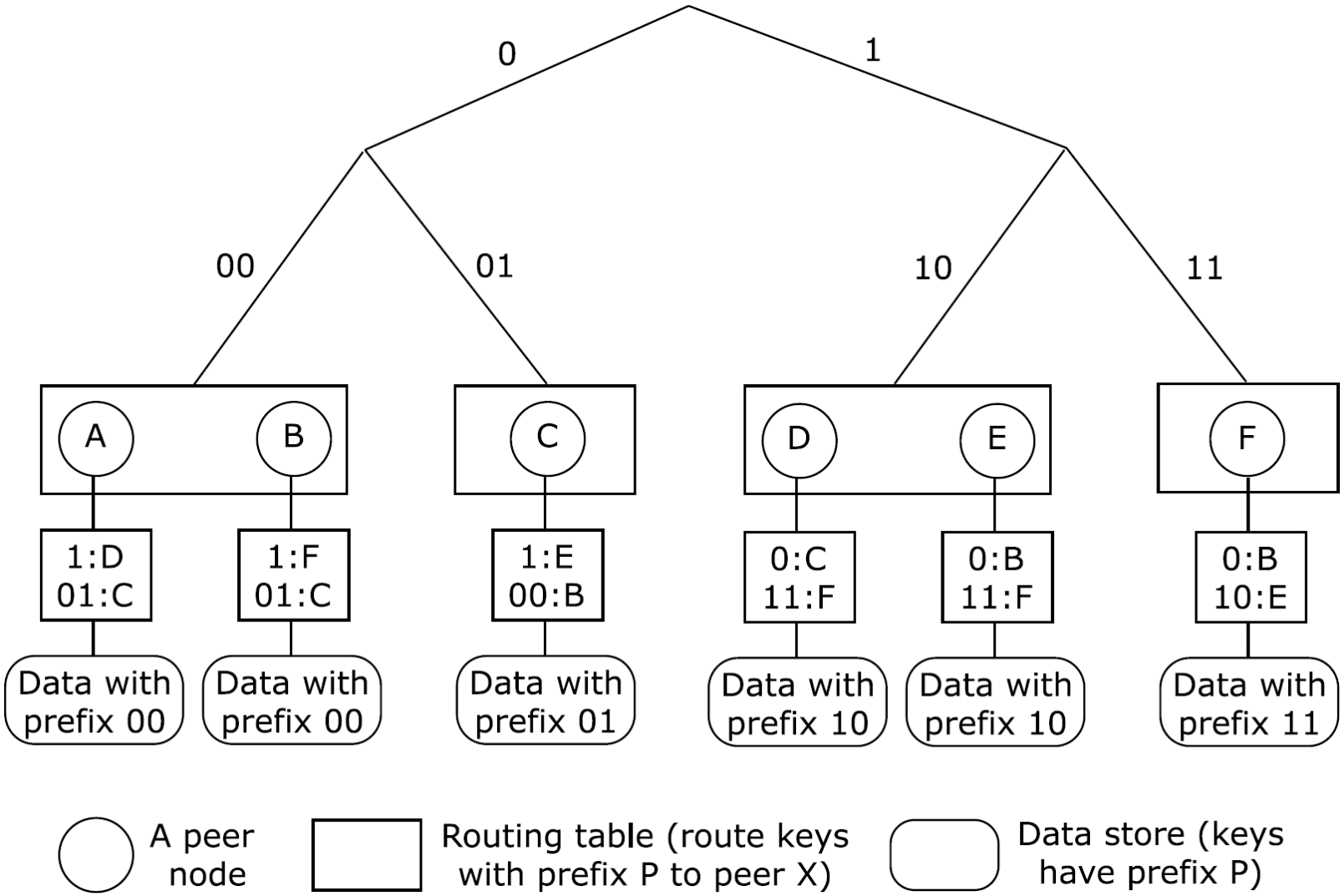
P-Grid I.

the idea:

- the P-Grid is based on a *virtual binary tree* structure in which each peer maintains a leaf node of the tree
- the system assigns each peer an identifier, which is the binary bit string representing the path from the root to the leaf node
- each peer is then responsible for all data items whose prefix is equal to the peer identifier
  - for fault-tolerance purposes, multiple peers can be assigned the same identifier
- for routing purposes, each peer further maintains a routing table

Tree based systems

P-Grid II.



Tree based systems

P-Grid III.

the routing/lookup mechanism:

- when a peer n receives a query having the key k, it checks, whether its identifier is a prefix of k
  - if yes, it searches its local storage to find the result
  - if no, the peer looks up its routing table to find a closer neighbor node to forward the query
- the maximum number of search steps is bounded by the height of the tree
  - $\Rightarrow$ the lookup performance is $O(log_2 N)$

Tree based systems

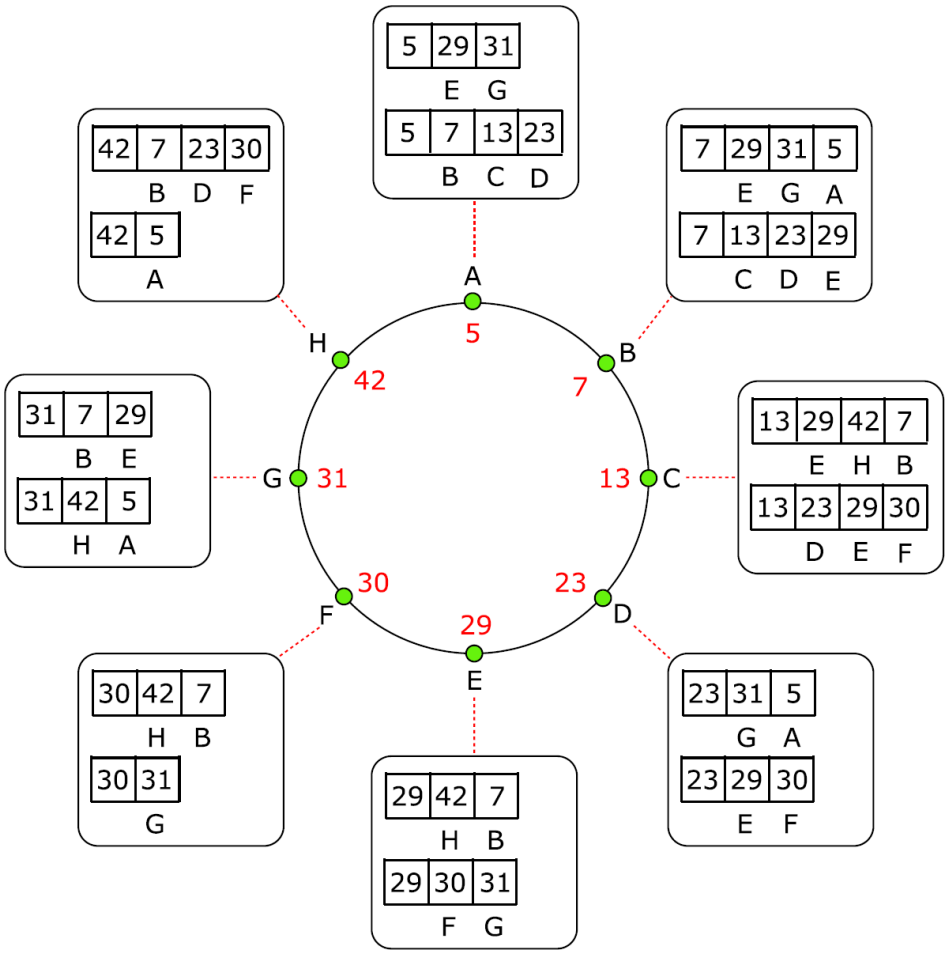
P-Tree I.

the idea:

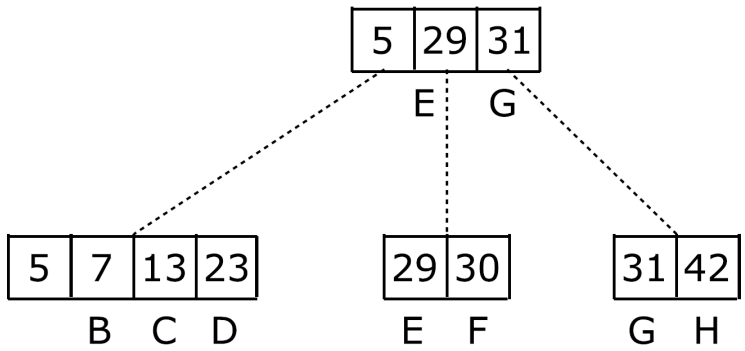
- in P-Grid, the balance of the tree structure cannot be guaranteed
  - P-Tree is based on a *virtual balanced B^+-Tree* built on top of a Chord ring
- each peer maintains:
  - a Chord node, which is a leaf node of the tree structure, and
  - a *semi-independent B^+-Tree*, which is a peer's view of a *fully independent B^+-Tree*
    - a *fully independent B^+-Tree* at a peer is a B^+-Tree, where the value stored at the peer is considered as the smallest value in the Chord ring
    - a *semi-independent B^+-Tree* contains all nodes in the leftmost root-to-leaf path of the corresponding fully independent B^+-Tree
    - to make it easy for maintenance, ranges of B^+-Tree nodes can be overlapped (see node C in the following figure)

Tree based systems

P-Tree II.



(a) Semi-independent B^+-Trees maintained at P-Tree nodes.



(b) The fully-independent B^+-Tree at node A.

Tree based systems

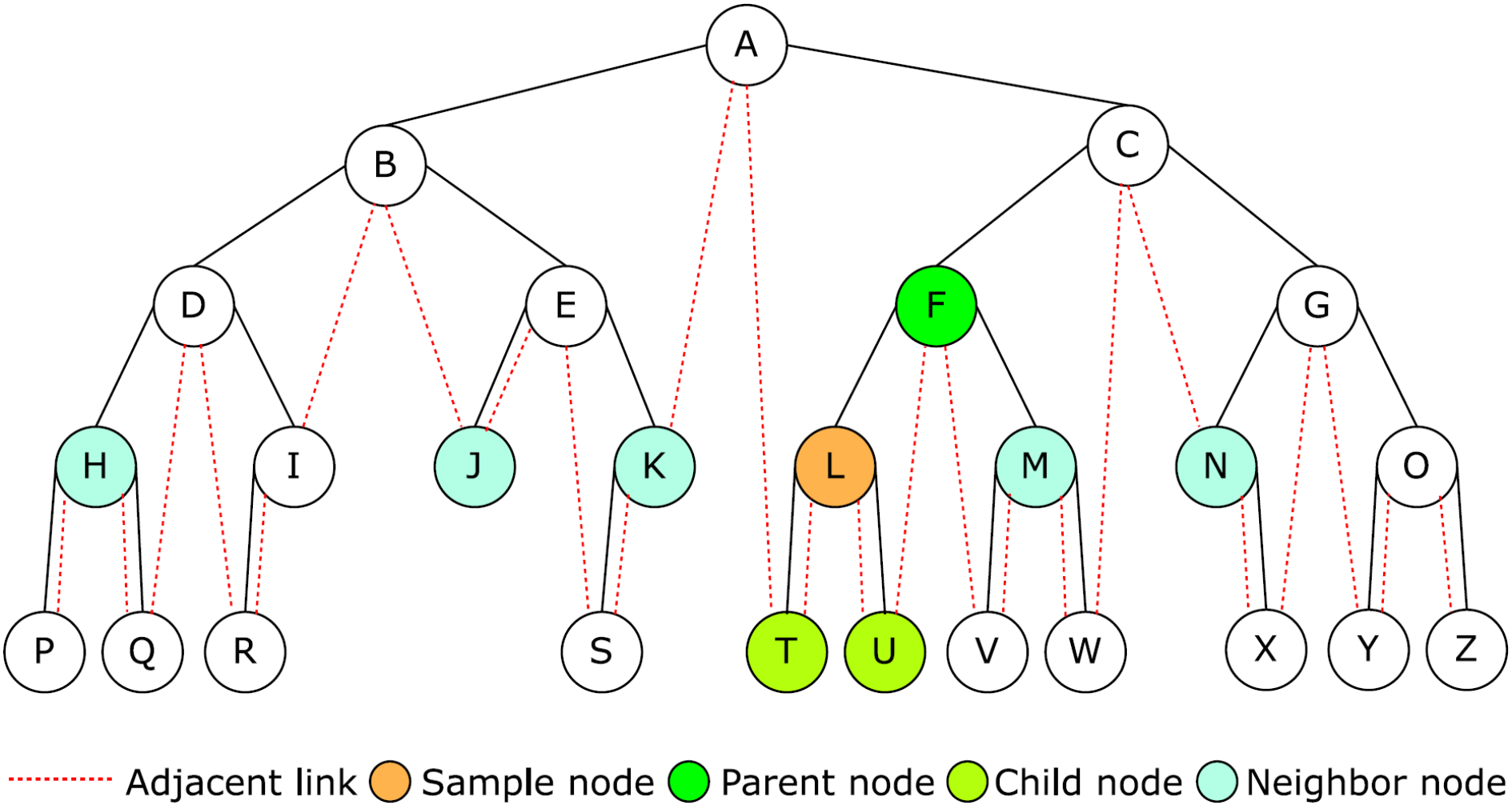
BATON I.

the idea:

- in comparison with standard tree-based structures, the BATON provides two main features:
  - data is stored at both leaf nodes and internal nodes
  - in addition to parent and child links, nodes in the BATON network also have *adjacent links* and *neighbor links*
    - *adjacent link* is used to connect a node to a node maintaining an adjacent range of values (adjacent to the range the node maintains)
    - *neighbor link* is used to connect a node with its neighbors (at the same level in the tree structure) having a distance 2^i, $i \geq 0$ from the node
    - the purpose of these links is to avoid the bottleneck problem at the root of the tree structure in query processing

Tree based systems

BATON II.



Tree based systems

BATON III.

lookup details:

- when a peer x receives a query:
  1. if the searched key falls into the range of values managed by x, it responds to the query
  2. otherwise, it forwards the query to the farthest neighbor that is nearer to but not overshooting the searched key
  3. if such a neighbor does not exist, x forwards the query to either a child (if it exists) or an adjacent node of x in the search direction

Tree based systems

BATON IV.

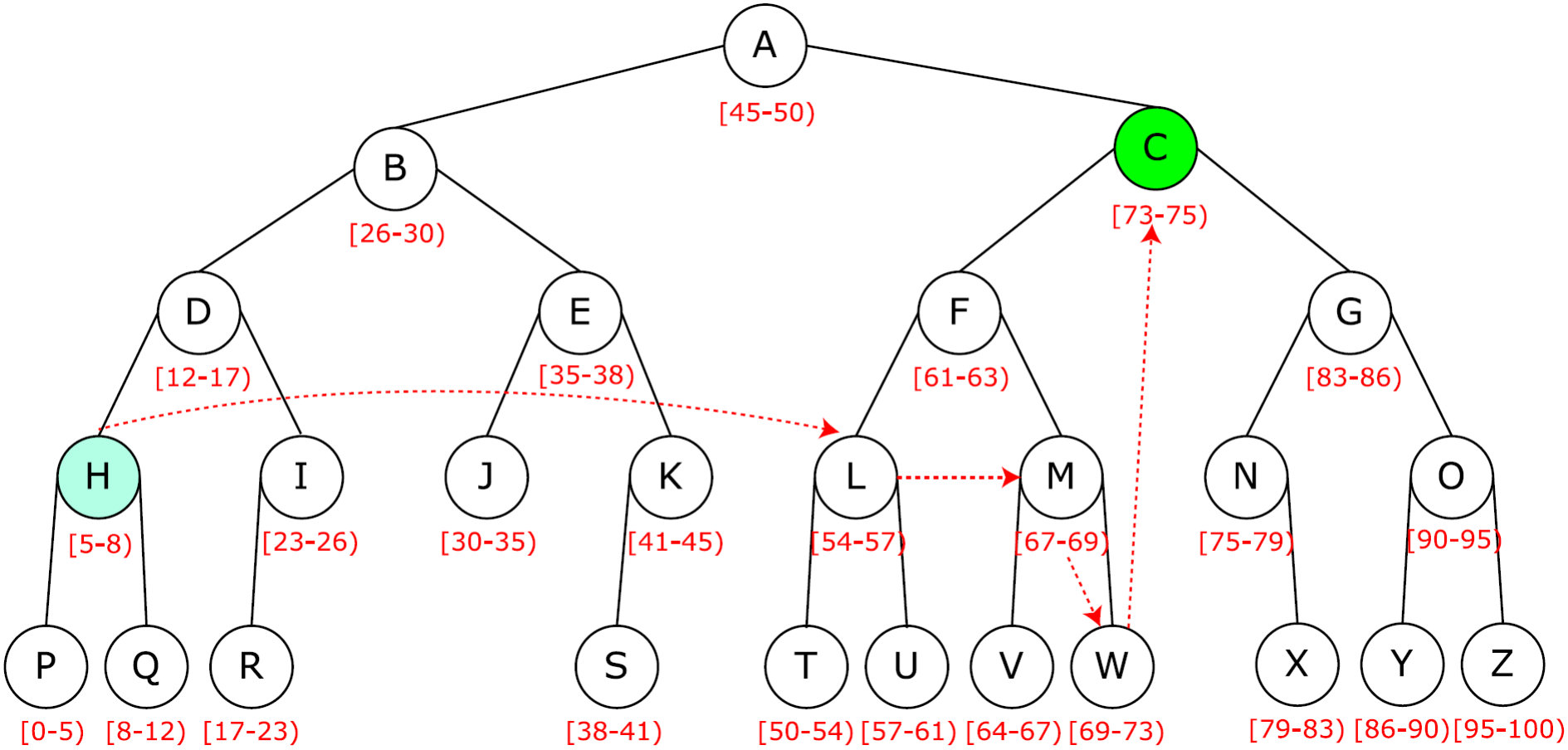


Figure: A lookup example in BATON: the node H wants to search for a data item (having the key 74) stored in the node C.

Routing in Hybrid P2P Networks

- hybrid P2P systems organize the peers into a hierarchical network
  - powerful peers (*superpeers*, *supernodes*) lie in a high level, and
  - common peers (also named *client peers*) lie in lower levels
  - each common peer belongs to a supernode and does not connect with any other common peer that does not belong to the same supernode
- the general routing scheme in hybrid P2P networks:
  1. a client peer sends a query to its supernode
  2. the supernode searches its directory to determine which client peer or supernode has the desired answers
  3. the query is sent to the supernode that may have the desired answers
     - it uses its directory of all its client peers to answer the query
  4. the IP address of the client peer having the desired answers is returned to the query peer
     - the query peer exchanges resources with that peer
- examples:
  - KaZaA, BestPeer, Edutella, etc.

Routing in Hybrid P2P Networks

Edutella

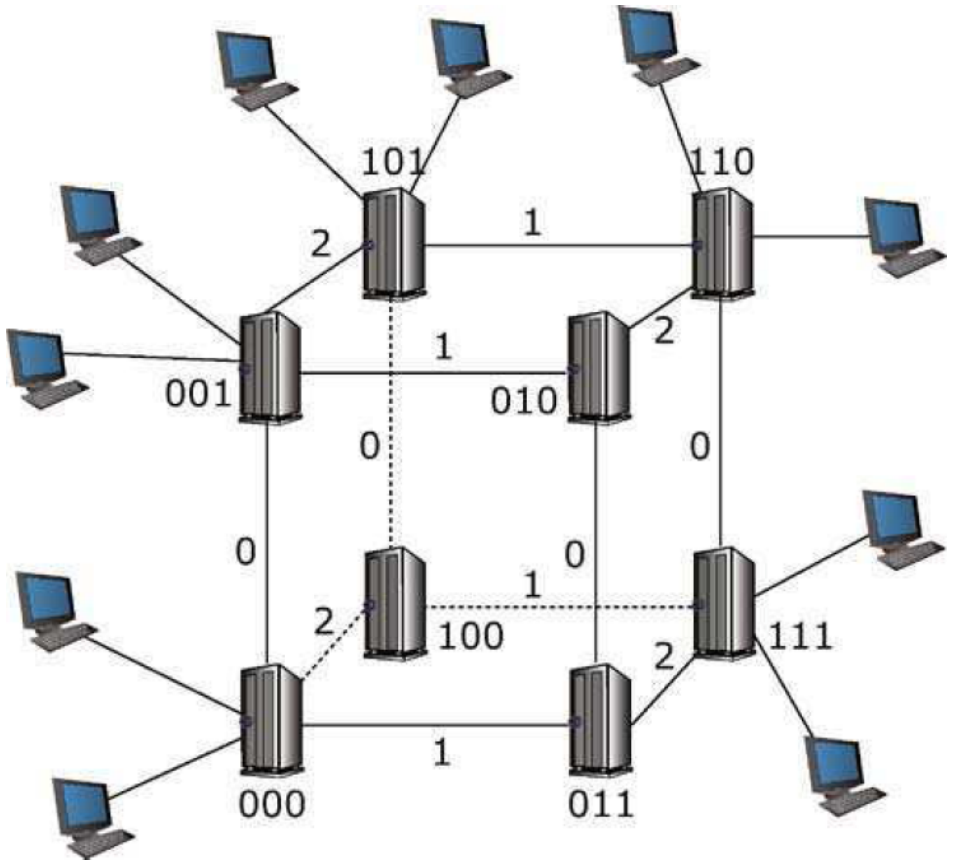


Figure: **The Edutella network structure.** A query routing in Edutella is first directed to superpeers in *HyperCuP network* (= HyperCube P2P network), where the suffix-based routing scheme could be employed.

Routing in Hybrid P2P Networks

Ultrapeers

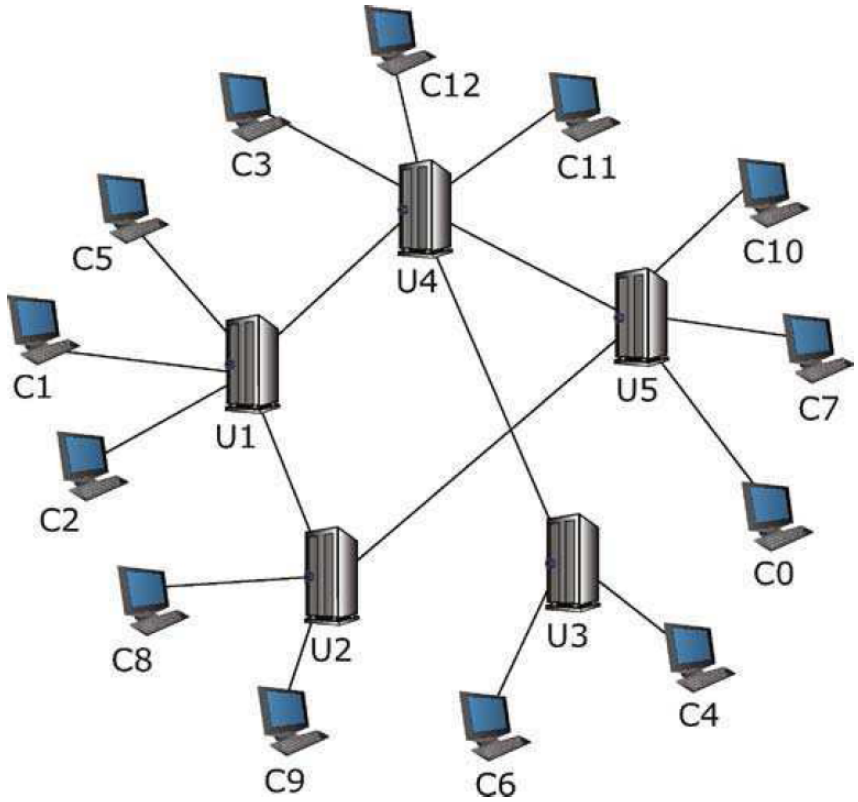


Figure: **The modified Gnutella network with ultrapeers.** Suppose that the resources requested by peer $C12$ are located at the peer $C9$: the peer $C12$ first requests its ultrapeer $U4$, then $U4$ floods the query to $U2$ via $U1$; $U1$ searches its reflector index and finds that $C9$ has the desired answers $\rightarrow$ it sends the IP address of $C9$ back to $C12$.

Routing in Hybrid P2P Networks

Structured Superpeers

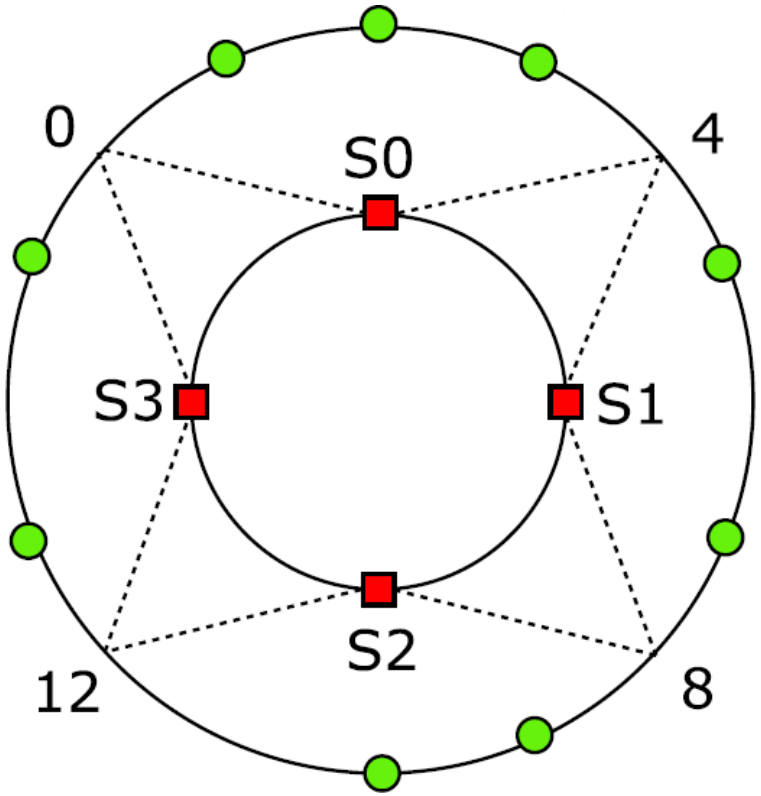


Figure: **The structured superpeers:** the superpeers $S0$, $S1$, $S2$, and $S3$ control ranges $(0, 4]$, $(4, 8]$, $(8, 12]$, and $(12, 0]$, respectively. If the peer $P1$ requests $key = 10$, it first sends the lookup key to $S0$; $S0$ relays the key to $S2$ (since $S2$ controls the range where the key belongs), which replies the query initiator with the IP of the relevant node storing the requested data

P2P Routing Conclusion

Structured vs. Unstructured P2P Networks Comparison

	structured P2P	unstructured P2P
routing	based on a routing table	flooding, random walk, ...
lookup possibilities	based on keys only	possibility to ask more compex queries
existing item is always found	yes	cannot be guaranteed
critical part	node join/disconnect	lookup/routing

P2P Routing Conclusion

Overview I.

System	Overlay network	Routing table	Routing method
Gnutella	Unstructured, Random topology	Random neighbors	Breadth First Search with Time-to-Live
FreeNet	Unstructured, Random topology	Random neighbors	Depth First Search with Time-to-Live
Chord	Structured, Ring topology	Neighbors at distances 2^i in the ring	Repeatedly jump to the farthest node in the routing table whose id is still less than the search key
CAN	Structured, Mesh topology	Neighbors at adjacent positions in the mesh	Repeatedly travel through the neighbor that is closer to the destination
Pastry & Tapestry	Structured, PRR tree topology	Neighbors sharing common prefix identifier at different levels	Repeatedly forward the message to the neighbor having the longest matching prefix identifier
Viceroy	Structured, butterfly topology	Five neighbors: one at the upper level, two at the lower level, and two at the same level	Three steps: going up, going down, and vicinity search
Crescendo	Structured, hierarchical ring topology	Chord-like neighbors at different ring levels	A combination of Chord-like routing and the routing between rings at different levels

P2P Routing Conclusion

Overview II.

System	Overlay network	Routing table	Routing method
Skip Graph	Structured, multiple linked lists topology	Neighbors sharing common prefix membership vector at different lengths	Travel from the highest to the lowest level of the list. At each level, jump to the neighbor closer to the destination if such a neighbor exists
SkipNet	Structured, hierarchical ring topology	Neighbors are predecessors and successors at different ring levels	Skip Graph-like routing, traveling from the highest to the lowest level of the ring.
P-Grid	Structured, binary tree topology	A neighbor at the other side of the tree rooted at each internal node from the root to the leaf	Travel from the root to the leaf. At each level, jump to the neighbor closer to the destination
P-Tree	Structured, a combination of a B^+-Tree and a Chord ring topology	Neighbors are nodes in the left-most root-to-leaf path of the B^+-Tree	Travel from the root to the leaf. At each level, jump to the neighbor closer to the destination
BATON	Structured, balanced tree topology	Neighbors are parent, children and Chord-like neighbors at the same level	If not having full routing tables, go to parent. Otherwise, go to the neighbor or the child closer to the destination
Edutella & Ultrapeers	Hybrid, a combination of structured and unstructured topology	Neighbors exist only at superpeer level. At client side, each client peer connects to a superpeer	A client peer always routes its requests to its superpeer while routing at supper peer level depends on the topology employed at that level

Lecture Overview I

P2P Information Sources

FI courses:

- PA128: Similarity Searching in Multimedia Data (prof. Zezula)

Literature:

- O. H. Vu at al. *Peer-to-Peer Computing: Principles and Applications.* Springer, 2010
- Milojicic et al. *Peer-to-Peer Computing.* HP Labs, 2002
- D. C. Verma. *Legitimate Applications of P2P Networks.* Wiley, 2004
- X. Shen, H. Yu, J. Buford, M. Akon. *Handbook of Peer-to-Peer Networking.* Spriger, 2010
- J. Buford, H. Yu, E. K. Lua. *P2P Networking and Applications.* Morgan Kaufmann, 2009