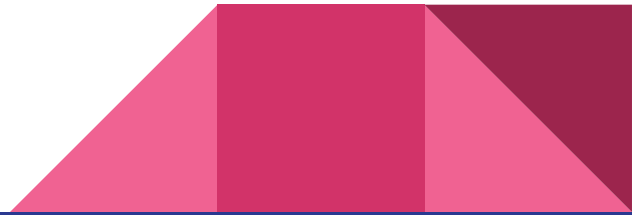
Peer-to-Peer

Lesson 10

Characteristics of P2P Networks

A peer-to-peer (P2P) network is an application-level overlay where resources such as files or multimedia content are shared directly across user machines. Every node is equal in capability and role, and each node can simultaneously behave as both a client and a server. Nodes communicate directly with each other without relying on centralised servers.

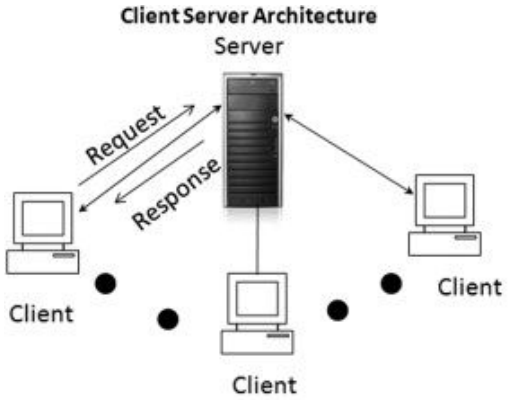
P2P systems offer high scalability because the addition of new peers increases both resource availability and capacity. They support low-cost expansion, entry and exit of nodes, and dynamic insertion or deletion of data objects. The continuous joining and leaving of nodes, called **churn**, is a natural behaviour in these networks. Well-designed P2P overlays aim to make churn transparent so ongoing operations are not disrupted. Another key property is **self-organisation**: these networks operate efficiently even without dedicated infrastructure or administrative authority.

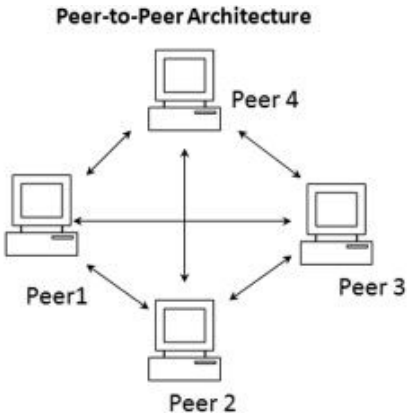


P2P vs Client-Server Networks

In the client-server model, a centralised server provides services while clients solely consume them. Roles are fixed and hierarchical. Communication patterns are predominantly one-to-many, and the server becomes a bottleneck and a single point of failure.

In P2P networks, all nodes are equal and able to serve and request data from one another. There is no central authority coordinating interactions. Nodes cooperate by directly exchanging information, which removes the need for specialised servers. This equality of roles enhances scalability because the system's capacity grows naturally as more peers join. Moreover, resource sharing depends on user machines rather than dedicated servers, allowing large-scale distribution of files or objects.

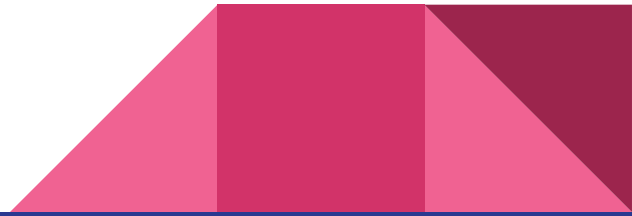




Churn in P2P Systems

Churn refers to the continuous entry and departure of peers from the P2P network, as well as dynamic insertion and deletion of objects. This behaviour arises because end-user devices connect and disconnect at will.

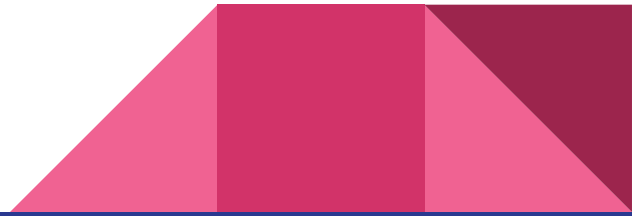
Churn can disrupt routing, lookup operations and availability of data. To minimise these disruptions, P2P overlays aim to make the effects of churn as transparent as possible through replication, dynamic routing adjustments and decentralised maintenance. The goal is to ensure stable performance even under high rates of peer turnover.



DNS Limitations and How P2P Overcomes Them

Traditional distributed systems rely on DNS to map host names to IP addresses. DNS depends on dedicated servers organised in a fixed hierarchy. Routing information must be manually configured to allow clients to navigate the DNS tree. DNS can only locate hosts or services, not arbitrary data objects stored on end-user devices. Hostnames must follow administrative structures, further limiting flexibility.

P2P networks eliminate these constraints by allowing direct location of arbitrary data objects without centralised servers. Lookup operations occur through decentralised routing within the overlay, making the system more flexible and scalable. Peers can locate content regardless of where it resides, which avoids the rigid administrative boundaries of DNS.



Examples of Well-Known P2P Networks

Several P2P networks support distributed file sharing across user machines.
Common examples include:

- **Napster** – early hybrid P2P system for music sharing.
- **Gnutella** – fully decentralised file-sharing network.
- **Freenet** – anonymous P2P platform for distributed file storage.
- **Pastry** – structured P2P overlay for routing and object location.
- **Chord** – structured P2P lookup protocol based on consistent hashing.
- **CAN (Content Addressable Network)** – structured P2P system using multi-dimensional coordinate space for object lookup.

These systems illustrate different P2P routing strategies, degrees of decentralisation and approaches to locating objects without central servers.

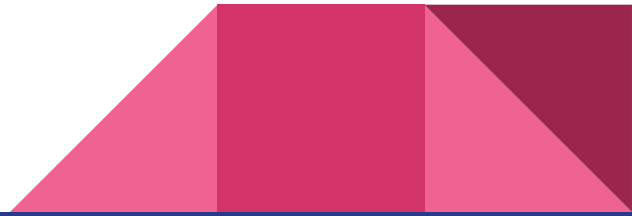
Self-Organisation and Scalability in P2P Networks

A major strength of P2P networks is their ability to self-organise without prior infrastructure. Nodes independently maintain routing state, adjust to churn and participate in data lookup and replication. Because each new peer contributes resources such as storage and CPU power, the total system capacity increases as the network grows.

This allows P2P overlays to scale to millions of users while maintaining acceptable performance. The decentralised nature avoids bottlenecks and ensures that no single machine becomes a performance limiter.

Advantages of P2P Overlays for Resource Sharing

P2P overlays allow flexible distribution of data stored across end users rather than relying on central servers. This makes them suitable for sharing large volumes of files, multimedia documents and other objects. They offer low entry and exit costs, support dynamic reconfiguration, and operate efficiently without administrative control. The decentralised architecture ensures robustness and fault-tolerance while accommodating unpredictable user behaviour.



Characteristics and Performance Features of P2P Systems

P2P systems are **self-organising**, meaning that nodes independently join and maintain the network without central administration. They operate under **distributed control**, with no single point managing decisions. All nodes have **role symmetry**, enabling any peer to act both as a server and client.

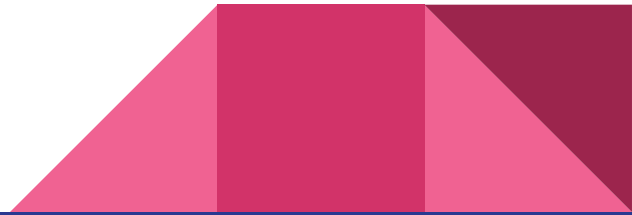
Features such as **anonymity** ensure that peers generally do not know each other's identities. Efficient **naming mechanisms** allow objects or nodes to be referenced and located. Security mechanisms such as authentication and trust are integrated into the overlay.

On the performance side, P2P networks provide **large aggregated resources**, including combined storage, CPU power, and bandwidth. They support **fast searching** for machines and objects, and they are **scalable**, improving performance as more nodes join. They handle **churn** efficiently through mechanisms that maintain stability even as nodes frequently enter or leave. Naming allows peers to select **geographically close servers**, while redundancy in storage and routing paths contributes to resilience.

Napster Architecture Overview

Napster was an early and widely used P2P file-sharing system. It employed a **server-mediated, centralised index** architecture. Clusters of servers maintained direct indices of the files available across all participating clients. Although file transfer occurred directly between users, the lookup stage relied on this centralised directory.

Each Napster server cluster stored and maintained metadata describing the files that clients could share. The servers enabled the discovery of peers who possessed the requested content and facilitated the initial connection setup.

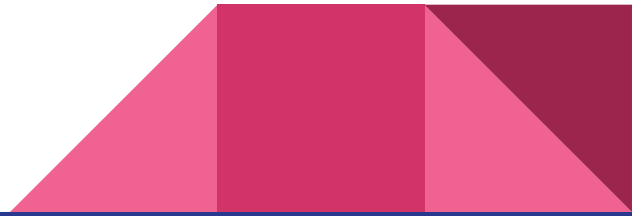


Information Stored by the Napster Central Server

The central server maintained a directory for every registered client. For each client, it stored:

- the client’s IP address and port,
- the offered bandwidth,
- information about the files the client allowed others to download.

This directory was essential for mapping a desired file to the specific hosts that stored it. It played the role of a large metadata index enabling rapid lookup.

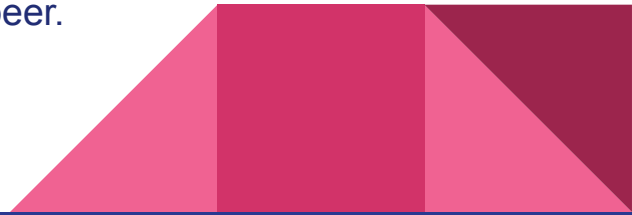


Napster File Search and Download Process

The basic operational steps were:

1. **Client contacts a meta-server**, which assigns a lightly loaded server from one of the nearby server clusters.
2. **Client connects to the assigned server** and submits its query along with its own identity.
3. **The server responds** with a list of users who possess the requested file, along with details about those users and their shared files.
4. **Client chooses a peer** from the provided list for downloading. The server supplies the necessary address information enabling the client to initiate a direct P2P connection.
5. The client and selected peer establish a direct transfer session to download the file.

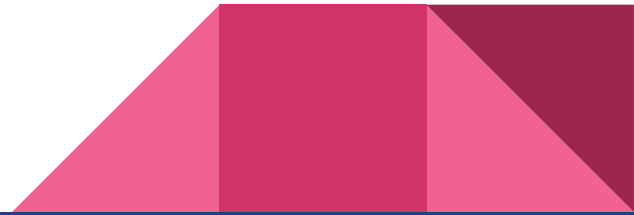
Thus, the server handles **file discovery**, while actual transfer is peer to peer.



Anonymity and Directory-Based Lookup in Napster

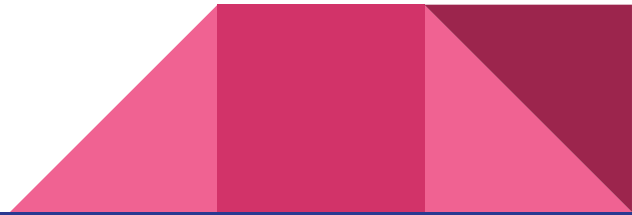
Users in Napster were generally **anonymous to each other**. They interacted through directory information rather than direct identity exchange. The central directory served as a mapping mechanism: it linked a file name to the IP address of peers offering that file. Clients relied on this mapping to determine from where the content should be downloaded.

This separation of lookup (via servers) and transfer (peer to peer) preserved anonymity between users and simplified file discovery.



Indexing Questions

1. Explain the three types of data indexing in P2P systems: centralised, local, and distributed.
2. What is distributed indexing in P2P networks? Why is it challenging?
3. Explain local indexing with its usage in P2P overlays.
4. Describe centralised indexing with examples such as DNS or Napster.
5. What is the role of indexing in P2P networks? Explain how indexing enables data independence.



Role of Data Indexing in P2P Systems

In a P2P network, each data object must be identified so that it can be located regardless of its physical storage location.
Indexing provides this identification and enables data independence, allowing applications to access objects without needing to know where they reside.

The slides classify indexing into three types:

- **Centralised indexing**
- **Local indexing**
- **Distributed indexing**

Centralised Indexing

Centralised indexing uses **one or a small number of central servers** to store index entries pointing to data located on many peers.
All lookup requests are sent to this central directory.

Examples:

- **DNS lookup** (maps names to IP addresses)
- **Early P2P systems like Napster**, which used a central directory server to maintain references to shared files

Characteristics:

- Simple and fast lookup
- Single point of failure
- Not scalable for large networks

Local Indexing

Local indexing requires **each peer to index only its own local data objects**.
If an object is remote, it must be searched for in the network.

- Local indexing is typically used in **unstructured overlays**
- Searching involves broadcast or flooding since peers do not maintain global knowledge

Characteristics:

- No central authority
- Very easy to maintain
- Works well in networks where data is spread unpredictably

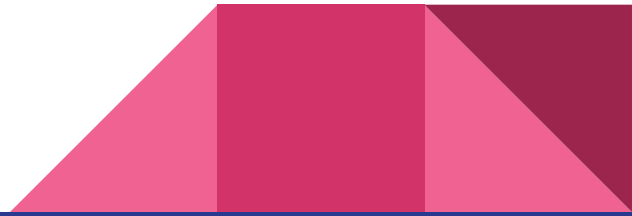
Distributed Indexing

Distributed indexing stores index entries **across many different peers**, rather than on a single server or only locally.

- Indexes are scattered across peers in the P2P overlay
- The overlay must provide a structure to access them
- This is the **most challenging** indexing approach
- A major mechanism used is the **Distributed Hash Table (DHT)**

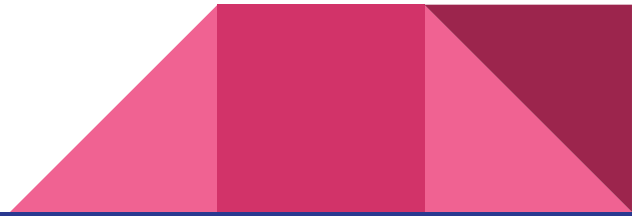
Features of DHTs:

- Each DHT differs in hash mapping strategy
- Different search algorithms
- Varying lookup diameter
- Fault-tolerance levels
- Ability to handle churn resiliently



Characteristics:

- Highly scalable
- Avoids central bottlenecks
- Complex to maintain under dynamic peer behaviour



Explain types of data indexing in P2P networks

Types of Data Indexing in P2P Networks

P2P systems identify data using indexing, which provides physical data independence and allows applications to access objects without knowing their exact storage location. The indexing mechanisms used in P2P overlays are:

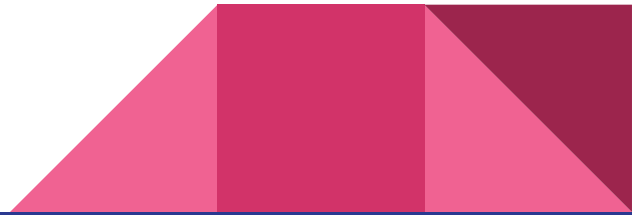
1. Centralised Indexing:
A central server (or a small set of servers) stores index references to objects located on many peers. Lookups are performed by querying this directory. DNS and early P2P systems like Napster used centralised directory lookup. This approach gives fast search but introduces a single point of failure.

2. Local Indexing:
Each peer maintains index entries only for its own local data objects. Remote objects must be searched in the network. This method is used in unstructured overlays, where no global structure exists. Local indexing requires minimal maintenance but can lead to expensive search operations.

3. Distributed Indexing:
Index entries are distributed across multiple peers in the overlay. A structured mechanism, often a Distributed Hash Table (DHT), is used to locate these scattered indexes. DHTs provide scalable lookup, fault-tolerance and resilience to churn, though distributed indexing is more complex to implement than other schemes.

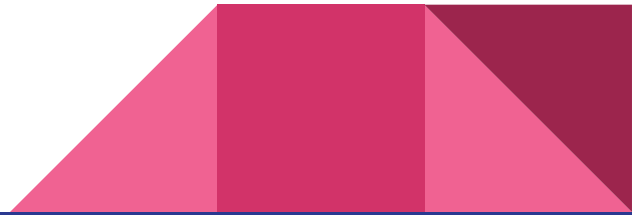
Overlays Questions

1. Explain structured overlays in P2P networks. How does deterministic mapping support fast lookup?
2. Describe the role of hash functions in structured overlays. Why are arbitrary queries difficult to support?
3. What are unstructured overlays? Explain their characteristics and the impact on search performance.
4. Discuss the advantages and disadvantages of unstructured overlays. When are they considered efficient?
5. Compare structured and unstructured overlays in terms of file placement, indexing, churn handling, and query performance.



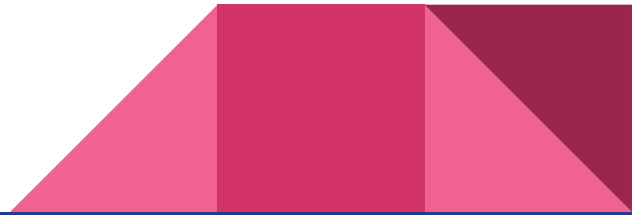
Explain structured overlays in P2P networks. How does deterministic mapping support fast lookup?

Structured overlays are peer-to-peer networks in which the topology follows a well-defined structure, and each file or data object is placed at a precise location determined by an algorithmic mapping. This deterministic mapping ensures that the location of a file is not random but computed from a specific attribute of the data, typically using a hash function. Because each key is mapped to a predictable position, the system can perform a fast and guaranteed lookup without relying on exhaustive search. These overlays operate as lookup systems, where the mapping from keys to values is handled using a hash-table-like interface built on top of the overlay's regular structure. As a result, queries can be resolved efficiently by routing them along the structured topology until they reach the node responsible for the requested key. This deterministic path reduces uncertainty and enables very low lookup time.



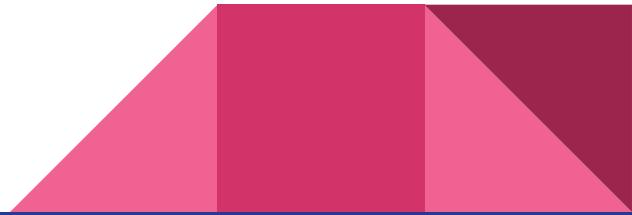
Describe the role of hash functions in structured overlays. Why are arbitrary queries difficult to support?

Hash functions play a central role in structured overlays by converting file attributes into numerical keys that determine the exact storage location of the data in the overlay. This mapping of keys to values ensures that the placement of objects is even and predictable, allowing the system to locate any file with high efficiency. However, this same deterministic mapping becomes a limitation when supporting more complex or semantic queries. Since the mapping is typically based on a single attribute, such as the file name, length, or a predetermined function, queries requiring relationships beyond this attribute cannot be processed directly. Range queries, attribute-based queries, or exact keyword searches cannot be executed naturally because the hash function destroys the semantic relationships between data objects. Therefore, only exact-key lookups are supported efficiently, while arbitrary queries require additional mechanisms or become impractical.



What are unstructured overlays? Explain their characteristics and the impact on search performance.

Unstructured overlays are peer-to-peer systems that do not impose any controlled or predefined structure on the network topology. Nodes connect in an ad-hoc manner, and file placement is not governed by a global mapping or algorithm. Each peer typically maintains an index only of its local data, which means that locating remote objects requires a network-wide search. Node joins and departures are simple to accommodate because the overlay does not depend on a rigid structure, and the topology adjusts naturally as peers join or leave. However, the absence of structured placement leads to high search overhead, since queries must be flooded or propagated widely across the network. Searches may take considerable time and, in some cases, fail altogether even when the object exists. As a result, unstructured overlays tend to suffer from high message overhead and variable performance during lookup operations.



Discuss the advantages and disadvantages of unstructured overlays. When are they considered efficient?

Unstructured overlays offer considerable flexibility because they allow complex queries, such as exact keyword queries, range queries, and attribute-based searches. These queries can capture the semantics of the data, enabling richer search capabilities than deterministic structured systems. Another advantage is their resilience to churn: rapid joining and departure of nodes does not degrade performance significantly because the overlay functions without a strict topology. Despite these benefits, unstructured overlays face major drawbacks. Searches may be slow or unreliable, requiring a large number of messages to locate data, and the overlay may still miss existing objects due to incomplete propagation. This creates scalability challenges for very large networks. Nevertheless, unstructured overlays become efficient when the system has some degree of data replication, when users accept best-effort search results, and when the network size is moderate enough that flooding-based search does not overwhelm the system.

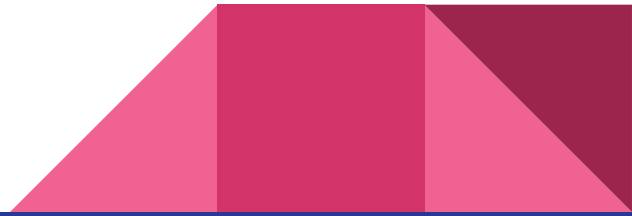
Compare structured and unstructured overlays in terms of file placement, indexing, churn handling, and query performance.

Structured and unstructured overlays differ fundamentally in how they organise data and manage lookup operations. In structured overlays, file placement is controlled by a deterministic algorithm, usually implemented via a hash function that maps keys to fixed locations in the overlay. This results in predictable indexing, fast lookup, and an efficient routing mechanism.

Unstructured overlays, by contrast, do not define where files are placed; each node stores whatever data it holds locally, and indexing is limited to these local objects. As a result, search operations may require network-wide exploration.

In terms of churn handling, unstructured overlays typically have an advantage because they can absorb frequent node arrivals and departures without disturbing a global structure. The topology adapts naturally, and no complex rebalancing is needed. Structured overlays require adjustments whenever peers join or leave, since keys and routing tables must be updated to preserve the deterministic mapping, though this process remains manageable in well-designed systems.

Query performance also differs significantly. Structured overlays excel at exact-key lookups, providing fast and guaranteed search performance. However, they struggle with semantic or range-based queries because the hash mapping removes meaningful relationships among data. Unstructured overlays, conversely, support a wide variety of complex queries but often incur high message overhead and unpredictable delay. Overall, structured overlays prioritise efficiency and determinism, while unstructured overlays emphasise flexibility and robustness under highly dynamic conditions.



Explain the iterative prisoner’s dilemma and its relevance to P2P system design.

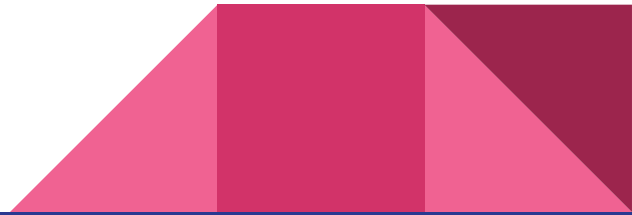
The iterative prisoner’s dilemma models repeated interactions between rational but selfish participants, where each round offers the choice to cooperate or betray. In a P2P system, this mirrors peers deciding whether to upload data (cooperate) or only download without contributing (betray). When the game is played only once, betrayal appears rational, but across repeated rounds, both participants benefit more by sustaining cooperation because each remembers prior behaviour and adjusts future responses. This repeated interaction pushes the system towards an equilibrium where cooperation emerges as the optimal long-term strategy. In the context of P2P networks, this outcome reflects the ideal solution in which peers continue sharing data, despite the temptation to free-ride. The model therefore demonstrates why practical P2P systems must encourage behaviour that aligns long-term benefits with cooperative participation.

Describe the tit-for-tat strategy and explain how BitTorrent applies it to handle the leeching problem.

Tit-for-tat is a simple reciprocal strategy in which a participant begins by cooperating and then repeats whatever action the other party took in the previous round. This encourages sustained cooperation because betrayal is met with an immediate reciprocal penalty in the next interaction. BitTorrent adopts this strategy by allowing uploads primarily to those peers who themselves upload data. A peer that contributes bandwidth and shares pieces of the file is rewarded with higher download rates, while a peer that refuses to upload is deprioritised and receives little or no data. By mirroring past behaviour, BitTorrent suppresses leeching, promotes fairness, and maintains high throughput in the swarm. The tit-for-tat approach thus enforces cooperation even among selfish peers, ensuring that the system operates efficiently.

What is trust or reputation management in P2P systems, and why is it required?

Trust or reputation management refers to the mechanisms used to estimate how reliable or cooperative a peer is within a P2P network. Since peers are autonomous and often anonymous, systems cannot assume that every node will behave honestly; some may provide corrupted data, leave abruptly, or fail to reciprocate uploads. The problem is intensified by high churn, where peers frequently join and leave, making it difficult to gauge reliability. Trust mechanisms therefore record or infer previous behaviour to estimate whether a peer is likely to act correctly. These approaches help users judge the quality of data sources, reduce exposure to malicious peers, and ensure that transactions take place between reliable participants. Without trust management, the system may degrade due to selfish or adversarial behaviour, reducing both performance and data quality.



Discuss the main challenges involved in designing trust management protocols in P2P systems.

Trust management in P2P systems faces several fundamental challenges. First, communication messages used to exchange trust information can be intercepted or forged, making the system vulnerable to man-in-the-middle attacks or Sybil attacks in which an adversary creates numerous fake identities to manipulate trust values. Second, because no peer has a complete global view of the system, it must rely on indirect information from other peers, which opens the possibility of false reports, collusion, and reputation distortion. Third, peers in P2P networks are highly transient, and trust values must adapt to continuous node arrivals and departures. This requires careful balancing so that trust estimates remain responsive to recent behaviour while still retaining meaningful historical evidence. Fourth, different peers may use different metrics to evaluate trust, such as accuracy of data, rate of cooperation, or consistency of responses, making aggregation and interpretation difficult. Finally, trust protocols add communication and computational overhead; designing them to be lightweight while still secure is a significant challenge. Together, these issues make trust management a complex but essential component of robust P2P system design.