Peer-to-Peer Systems: An Introduction

Hui Chen [a]

[a]CUNY Brooklyn College

December 3, 2025

Client-Server vs. Peer-to-Peer

- Client-Server Architecture
  - Centralized server
  - Clients request services from the server
  - Examples: Web servers, email servers
- Peer-to-Peer (P2P) Architecture
  - Decentralized network
  - Each node (peer) acts as both client and server
  - Examples: File sharing networks, blockchain networks

Characteristics of P2P Architecture

- Self-organizing
- Decentralization
- Scalability
- Resource Sharing
- Fault Tolerance
- Dynamic Network Topology

Examples of P2P Applications

- File Sharing
  - BitTorrent
  - Gnutella
- Communication
  - Skype
  - WhatsApp (some features)
- Blockchain and Cryptocurrencies
  - Bitcoin
  - Ethereum

Peer-to-Peer File Sharing Example: Motivation

What about client-server file sharing?

- Client request files, server responds with the data
- Disadvantage:
  - Single point of failure
  - Scalability issues with many clients
- Strategies to address disadvantages:
  - Use multiple servers, place them strategically close to clients

Any problems?

Peer-to-Peer File Sharing Example: Solution

- Distributed architecture to the "fullest extent."
- Each node has client and server logic.
- A peer downloads part of the file from another peer
- The client also servers parts of the file to other peers.
- Scalability? Failure of nodes or network?

Peer-to-Peer File Sharing Example: Discussion

- How does a node find other nodes that have the desired file?
- How to ensure data integrity and security in a decentralized network?
- What if users (peers) aren't willing to upload?

Peer-to-Peer File Sharing Example: BitTorrent

- Create a .torrent file: metadata about the file to be shared (filename, length, data about pieces that make up the file, URL of tracker).
- Have a tracker. A server that knows the identities of peers sharing the file in a file transfer.
- To download a file:
  1. A peer downloads the .torrent file.
  2. The peer contacts the tracker to get a list of peers sharing the file.
  3. The peer connects to these other peers, begins to transfer blocks of the file.
  4. Seeders: peers that have the complete file and continue to share it.

Peer-to-Peer File Sharing Example: BitTorrent Incentive Mechanism

To encourage sharing: peers do not get data unless they also upload data.

In practice, algorithmically implemented as:

- The system operates in rounds (typically every 10 seconds).
- In round n, some peers upload blocks to peer X.
- In round n+1, Peer X will send blocks to the peers that uploaded the most in round n, typically top 4 peers.
- To get started, each peer reserves some small amount of bandwidth to give away freely.

Peer-to-Peer File Sharing Example: BitTorrent Tracker Problem

The tracker is a single point of failure.

- If the tracker goes down, peers cannot find each other.
- Solution: Distributed Hash Table (DHT)
  - A decentralized system where each peer maintains a portion of the hash table.
  - Peers can look up other peers sharing a file without relying on a central tracker.

What is a DHT?

Key-value store distributed across peers; keys are hash values serving as content IDs.

```
key1, value1
key2, value2
key3, value3
...
```

Provide Key-Value store interface:

```
put(key,item)
get(key)
```

DHT Designs

Notable examples:

- Chord (Stoica et al. 2001)
  - Consistent hashing to assign keys to nodes.
  - Each node maintains a finger table for efficient routing.
- Kademlia (Maymounkov and Mazieres 2002)
  - XOR metric for distance between keys and nodes.
  - Uses k-buckets to store contact information of peers.
- Pastry (Rowstron and Druschel 2001)
  - Prefix-based routing.
  - Each node maintains a routing table, leaf set, and neighborhood set.

DHT Lookup and Routing

- Each peer maintains neighbor pointers (finger table or k-buckets).
- Lookup forwards to the neighbor "closest" to the target key.
- Time complexity: $O(\log n)$ hops under normal conditions.
- Robustness via replication and periodic refresh to handle churn.

Chord: Identifier Circle and Finger Table

Identifier Circle:

- Nodes and keys mapped to m-bit identifier space (hash values): $[0, 2^m - 1]$
- Form a logical ring
- Key k stored at successor node: first node $\geq k$

Example: $2^m = 2^6 = 64$ positions

- Nodes 1 - 8: 8, 14, 21, 32, 38, 42, 48, 51 (Keys = Hash Values = IDs)
- Key 54 stored at node 8 (successor)

Finger Table:

- Node n maintains m entries
- Entry i: successor of $(n + 2^{i-1}) \bmod 2^m$
- Enables $O(\log n)$ lookup

Node 8's finger table:

i	start	succ
1	9	14 (1st node $\geq$ start=9)
2	10	14
3	12	14
4	16	21
5	24	32
6	40	42

Chord: Node Join and Updates

When node n joins:

1. **Find position:** Contact a known node, use lookup to find n's successor
2. **Initialize finger table:** Populate n's finger table by querying existing nodes
3. **Update successors:** Notify n's successor and predecessor
4. **Transfer keys:** Successor transfers keys in range $(predecessor, n]$ to n
5. **Update other finger tables:** Nodes whose finger tables should point to n are updated

Example: Node 26 joins between 21 and 32

- Node 26's successor: 32, predecessor: 21
- Keys 22-26 transfer from node 32 to node 26
- Node 21's finger entry 4 (start=24) updates: $32 \rightarrow 26$
- Node 8's finger entry 5 (start=24) updates: $32 \rightarrow 26$

Chord: Stabilization

Churn: The dynamic process of nodes joining and leaving the network, which can disrupt routing and key storage.

Periodic stabilization protocol runs to repair finger tables after churn:

- Nodes periodically verify and update their successors and predecessors
- Finger table entries are refreshed to reflect current network state
- Failed nodes are detected and removed from routing tables
- Ensures $O(\log n)$ lookup complexity is maintained despite network changes

Kademlia: XOR Distance Metric

Key idea: Use XOR to measure distance between node IDs and keys

Distance between node a and node b: $d(a, b) = a \oplus b$

Properties of XOR distance:

- $d(a, a) = 0$ (distance to self is zero)
- $d(a, b) > 0$ if $a \neq b$ (positive distance)
- $d(a, b) = d(b, a)$ (symmetric)
- $d(a, b) + d(b, c) \geq d(a, c)$ (triangle inequality)
- Unidirectional: for any point x and distance $\Delta > 0$, there's exactly one point y such that $d(x, y) = \Delta$

Example: $m = 4$ bit IDs

- $d(1010_2, 1100_2) = 0110_2 = 6$
- $d(0011_2, 1011_2) = 1000_2 = 8$

Kademlia: Node Lookup Using XOR Distance

Routing: Forward query to node closest (smallest XOR distance) to target key

Example: Find node responsible for key 1101_2 (13)
Starting at node 0010_2 (2):

- Distance: $d(2, 13) = 0010_2 \oplus 1101_2 = 1111_2 = 15$
- Node 2 checks its k-buckets and finds: nodes 6, 10, 14
- Calculate distances from these known nodes to target:
  - $d(6, 13) = 0110_2 \oplus 1101_2 = 1011_2 = 11$
  - $d(10, 13) = 1010_2 \oplus 1101_2 = 0111_2 = 7$
  - $d(14, 13) = 1110_2 \oplus 1101_2 = 0011_2 = 3 \leftarrow$ closest!
- Forward query to node 14, which repeats the process with its own k-buckets

Each hop halves the distance $\Rightarrow O(\log n)$ hops

Kademlia: k-Buckets

k-bucket: List of up to k contacts (typically $k = 20$) for nodes at certain distance ranges

Structure: Each node maintains m k-buckets (for m-bit ID space)

- Bucket i ($0 \le i < m$): stores contacts for nodes at distance $[2^i, 2^{i+1})$
- Bucket i covers nodes whose IDs differ in the $(i + 1)$-th bit from the node's ID

Example: Node 0010_2 with $m = 4$:

- Bucket 0: nodes at distance [1,2): XOR $\in \{0001_2\} \rightarrow$ nodes $\{0011_2\}$
- Bucket 1: nodes at distance [2,4): XOR $\in \{0010_2, 0011_2\} \rightarrow$ nodes $\{0000_2, 0001_2\}$
- Bucket 2: nodes at distance [4,8): XOR $\in \{01??_2\} \rightarrow$ nodes $\{0110_2, 0111_2, ...\}$
- Bucket 3: nodes at distance [8,16): XOR $\in \{1???_2\} \rightarrow$ nodes $\{1010_2, 1101_2, ...\}$

Kademlia: k-Bucket Management

Bucket updates:

- When node n learns about node x:
  1. Calculate $d(n, x)$ to determine bucket i
  2. If bucket i has $< k$ entries: add x
  3. If bucket i is full: ping least-recently-seen node
     - If responds: move to tail (keep), discard x
     - If no response: evict, add x
- **Rationale:** Long-lived nodes more likely to remain online (prefer old contacts)

Lookup process:

- Query α closest nodes in parallel (typically $\alpha = 3$)
- Iteratively query closer nodes until finding k closest nodes to target
- More robust than Chord: multiple paths, redundancy

Kademlia: Node Join Process

When a new node n joins with ID id_n:

1. **Bootstrap:** Contact at least one known node b in the network
2. **Self-lookup:** Perform a node lookup for id_n (its own ID)
   - Discovers nodes close to itself
   - Populates its k-buckets with these contacts
3. **Bucket refresh:** For each empty or sparse k-bucket i:
   - Generate a random ID in bucket i's range
   - Perform lookup to find nodes in that distance range
4. **Announce presence:** Node n is now in other nodes' k-buckets through:
   - Responses to lookups (nodes add n when they receive messages)
   - Periodic republishing of keys it becomes responsible for

Key difference from Chord: No explicit finger table updates; nodes learn about n organically through queries and responses

Kademlia: Handling Node Failures and Churn

No explicit stabilization protocol like Chord

Instead, Kademlia uses passive techniques:

When a node shuts down or fails:

- **Redundancy:** Each k-bucket stores up to k contacts (typically $k = 20$)
  - Multiple nodes at similar distances provide backup routes
  - If one node fails, others in the same bucket can still route queries
- **Lazy eviction:** Failed nodes removed only when:
  - They don't respond to pings during bucket updates
  - A new contact needs to be added to a full bucket
- **Data replication:** Keys stored on k closest nodes (not just one)
  - If a node fails, others still have the data
  - Periodic republishing (every hour) ensures availability
- **Iterative lookups:** Query multiple nodes in parallel ($\alpha = 3$)
  - If one route fails, others succeed
  - More robust than Chord's sequential routing

DHT Considerations

- Consistency: eventual; updates propagate via refresh and republishing.
- Security: Sybil/Eclipse risks; mitigations include diversity, rate limits, signatures.
- Practicality: NAT traversal, bootstrapping peers, throttling to avoid overload.
- Trade-offs: less central control, more resilience and scalability.

Blockchain: A P2P Distributed Ledger

Blockchain: A distributed, immutable ledger maintained by a P2P network

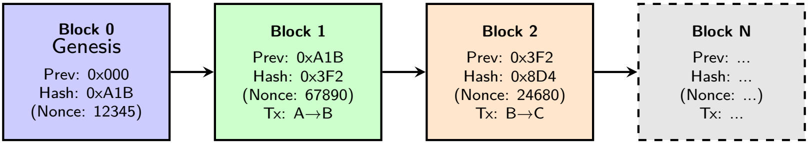
Key components:

- **Block:** Container of transactions with timestamp, hash of previous block
- **Chain:** Linked sequence of blocks forming transaction history
- **Distributed ledger:** Every node maintains a full or partial copy
- **Consensus:** Nodes agree on the valid state of the ledger

P2P characteristics:

- No central authority; all nodes are equal (in theory)
- Nodes broadcast transactions and blocks to peers
- Self-organizing: nodes join/leave freely
- Fault tolerant: system continues if some nodes fail

Blockchain Structure: Visual Representation



Key properties:

- Each block contains hash of previous block (cryptographic linkage)
- Changing any block invalidates all subsequent blocks
- Immutability through chained hashing

Note: Not all blockchains don't use nonces (e.g., Proof-of-Stack Blockchains); validators are selected by stake

Several Blockchain Terms

Genesis Block:

- The very first block in a blockchain (Block 0)
- Has no previous block to reference (Prev hash = 0x000...)
- Hard coded into the blockchain protocol
- Example: Bitcoin's genesis block

Nonce (Number used Once):

- A random number used in **Proof of Work (PoW)** mining
- Miners (PoW-specific) repeatedly change the nonce to find a hash meeting difficulty target
- Example: Find hash < 0x0000FFFF... (leading zeros indicate difficulty)
- Process: Try nonce = 1, 2, 3, ... until valid hash found
- Valid block: hash(prev_hash + transactions + nonce) meets target

Example (PoW): If target requires 4 leading zeros, miner tries millions of nonces until finding one where hash = 0x0000A3F2...

Bitcoin: Decentralized Cryptocurrency

Bitcoin: First successful cryptocurrency using blockchain technology (Nakamoto 2008)

How it works as a P2P system:

1. **Transaction creation:** User creates and signs transaction
2. **Broadcasting:** Transaction broadcast to peer nodes
3. **Validation:** Nodes verify signature and sufficient balance
4. **Mining (PoW-specific):** Miners compete to add block of transactions to chain
   - Solve computational puzzle (Proof of Work)
   - First to solve broadcasts new block to network
   - **Note:** PoS blockchains use validators instead of miners
5. **Consensus:** Nodes accept longest valid chain as truth
6. **Reward:** Miner receives newly minted coins + transaction fees

P2P discovery: Bitcoin nodes find peers via DNS seeds, hardcoded addresses, gossip protocol

Blockchain P2P Network Architecture

Network topology: Unstructured P2P (unlike DHTs)

Node types:

- **Full nodes:** Store entire blockchain, validate all transactions
- **Light nodes (SPV):** Store block headers only, verify via Merkle proofs
- **Mining nodes:** Participate in block creation
- **Archive nodes:** Store full history including pruned data

Communication:

- Each node maintains connections to 8-125 peers
- Gossip protocol: broadcast transactions/blocks to neighbors
- No routing tables (unlike DHT)
- Flooding for propagation

Comparison to DHT:

- DHT: Structured, $O(\log n)$ routing, key lookup
- Blockchain: Unstructured, flooding, consensus on global state

Mining Algorithm: Solving the Computational Puzzle

The Puzzle: Find a nonce such that the block hash meets a difficulty target

Algorithm:

1. Collect pending transactions into a candidate block
2. Set initial nonce = 0
3. Compute: $h = \text{SHA256}(\text{prev\_hash}|\text{transactions}|\text{nonce})$
4. Check if $h <$ target (equivalently: h has required leading zeros)
   - If yes: broadcast block, claim reward
   - If no: increment nonce, go to step 3

Example (simplified): Target = 0x0000FFFF... (4 leading zeros)

```
nonce=1: hash=0x3A5F... (invalid)
nonce=2: hash=0x9B2E... (invalid)
...
nonce=456789: hash=0x0000A3F2... (valid!)
```

Candidate Block

Pending transactions:

- **Users create transactions:** Alice wants to send 5 BTC to Bob
- **Wallet signs transaction:** Uses Alice's private key to authorize
- **Broadcast to P2P network:** Transaction sent to connected peers
- **Gossip propagation:** Each node forwards to its neighbors
- **Miners receive:** All nodes (including miners) receive the transaction

Building the candidate block:

1. **Validation:** Verify signature, sufficient balance, no double-spending
2. **Mempool storage:** Store valid pending transactions in memory pool
3. **Select transactions:** Choose from mempool (typically highest fee first)
4. **Add coinbase:** Create transaction awarding mining reward to miner
5. **Assemble header:** prev_hash, merkle_root, timestamp, target, nonce=0

Result: Candidate block ready for mining

Consensus Mechanisms in Blockchain P2P

Challenge: How do peers agree on transaction order without central authority?

Proof of Work (PoW): Bitcoin, Ethereum (pre-2022)

- Miners solve cryptographic puzzle to create block
- Energy intensive; prevents Sybil attacks (costly to gain majority)
- Longest chain wins in case of forks

Proof of Stake (PoS): Ethereum (post-2022)

- Validators chosen based on stake (coins held)
- More energy efficient; validators risk losing stake if dishonest
- Various selection mechanisms (random, committee-based)

Other mechanisms:

- Practical Byzantine Fault Tolerance (PBFT): voting among known validators
- Delegated PoS: token holders vote for validators

Blockchain P2P: Challenges and Solutions

- **Scalability:**
  - Problem: Limited throughput (Bitcoin 7 tx/sec, Ethereum 15 tx/sec)
  - Solutions: Layer 2 (Lightning Network), sharding, off-chain processing
- **Network partitions:**
  - Problem: Network splits create forks
  - Solution: Longest chain rule; eventually reconciles
- **51% attacks:**
  - Problem: Entity controlling majority can manipulate chain
  - Mitigation: High cost of acquiring majority (PoW/PoS)
- **Privacy:**
  - Problem: Transactions publicly visible
  - Solutions: Mixing services, privacy coins (Monero, Zcash)
- **Storage:**
  - Problem: Blockchain grows indefinitely (Bitcoin 500GB)
  - Solutions: Pruning, light clients, state channels

Blockchain vs. Traditional P2P Systems

Feature	DHT (Chord/Kademlia)	Blockchain
Topology	Structured	Unstructured
Routing	$O(\log n)$	Flooding/gossip
Data model	Key-value store	Append-only ledger
Consistency	Eventual	Strong (consensus)
Goal	Efficient lookup	Agreement on state
Use case	File sharing, discovery	Cryptocurrency, trust
Mutability	Values can change	Immutable history
Incentives	Altruism/reciprocity	Economic rewards

Similarities:

- Decentralized, no single point of failure
- Self-organizing, dynamic membership
- Fault tolerant through replication

Summary

- **P2P characteristics:** Decentralized, self-organizing, fault-tolerant, scalable
- **BitTorrent:** File sharing with tit-for-tat incentives; tracker or DHT for discovery
- **DHT (Chord/Kademlia):** Structured P2P with $O(\log n)$ lookup
  - Chord: Consistent hashing ring, finger tables, explicit stabilization
  - Kademlia: XOR metric, k-buckets, passive fault tolerance through redundancy
- **Blockchain:** Unstructured P2P with distributed ledger and consensus
  - Bitcoin: PoW consensus, economic incentives, immutable transaction history
  - Different topology and goals compared to DHT (agreement vs. lookup)

References I

- Maymounkov, Petar and David Mazieres (2002). “Kademlia: A peer-to-peer information system based on the xor metric”. In: *International workshop on peer-to-peer systems*. Springer, pp. 53–65.
- Nakamoto, Satoshi (2008). “Bitcoin: A Peer-to-Peer Electronic Cash System”. In: URL: https://bitcoin.org/bitcoin.pdf.
- Rowstron, Antony and Peter Druschel (2001). “Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems”. In: *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, pp. 329–350.
- Stoica, Ion et al. (2001). “Chord: A scalable peer-to-peer lookup service for internet applications”. In: *ACM SIGCOMM computer communication review* 31.4, pp. 149–160.