Distributed Systems

15-440 / 15-640
Fall 2017

P2P and Blockchains
Motivation for P2P: The Scaling Problem

- Millions of clients \(\Rightarrow\) server and network meltdown
P2P System

- Leverage the resources of client machines (peers)
  - Computation, storage, bandwidth
Peer-to-Peer Networks

- Typically each member stores/provides access to content
- Basically a replication system for files
  - Always a tradeoff between possible location of files and searching difficulty
- Challenge 1: searching / lookup (files can be anywhere)
  - Dynamic member list makes it more difficult
- Challenge 2: fairness (everyone gets rightful share and everyone contributes)
  - Contribution to P2P system must be incentivized.
- Challenge 3: trust (downloading files from unknown sources)
  - Doesn’t help that we associate P2P with illegal downloads
Topics Today

P2P Challenge 1: Efficient lookups
   Centralized
   Flooding
   BitTorrent
   Hashing-Based (Chord)

P2P Challenge 2: Fairness
   BitTorrent

P2P Challenge 3: Trust
   BitTorrent
   Blockchains
The Lookup Problem

Key="title"
Value=MP3 data
Publisher

Client
Lookup("title")
Searching

• Needles vs. Haystacks
  • Searching for top 40, or an obscure punk track from 1981 that nobody’s heard of?

• Search expressiveness
  • Whole word? Regular expressions? File names? Attributes? Whole-text search?
    • (e.g., p2p gnutella or p2p google?)
Framework

• Common Primitives:
  • **Join**: how do I begin participating?
  • **Publish**: how do I advertise my file?
  • **Search**: how do I find a file?
  • **Fetch**: how do I retrieve a file?
Topics Today

P2P Challenge 1: Efficient lookups
  - Centralized
  - Flooding
  - BitTorrent
  - Hashing-Based (Chord)

P2P Challenge 2: Fairness
  - BitTorrent

P2P Challenge 3: Trust
  - BitTorrent
  - Blockchains
Napster: Overview

- Centralized Database:
  - **Join**: on startup, client contacts central server
  - **Publish**: reports list of files to central server
  - **Search**: query the server => return someone that stores the requested file
  - **Fetch**: get the file directly from peer
Napster: Discussion

- **Pros:**
  - Simple
  - Search scope is $O(1)$
  - Controllable (pro or con?)

- **Cons:**
  - Server maintains $O(N)$ State
  - Server does all processing
  - Single point of failure
Topics Today

P2P Challenge 1: Efficient lookups
- Centralized
- Flooding
- BitTorrent
- Hashing-Based (Chord)

P2P Challenge 2: Fairness
- BitTorrent

P2P Challenge 3: Trust
- BitTorrent
- Blockchains
Gnutella: Overview

- Query Flooding:
  - **Join**: on startup, client contacts a few other nodes; these become its “neighbors” - word of mouth/caching
  - **Publish**: no need
  - **Search**: ask neighbors, who ask their neighbors, and so on... when/if found, reply to sender.
    - TTL limits propagation
  - **Fetch**: get the file directly from peer
Gnutella: Discussion

- **Pros:**
  - Decentralized
  - Search cost distributed
  - Processing @ each node permits powerful search semantics

- **Cons:**
  - Search scope is $O(N)$
  - Search time is $O(???)$
  - Nodes leave often, network unstable
  - For scalability, does NOT search every node. May have to re-issue query later
Flooding: Gnutella, Kazaa

- Modifies the Gnutella protocol into two-level hierarchy
  - Hybrid of Gnutella and Napster
- Supernodes
  - Nodes that have better connection to Internet
  - Act as temporary indexing servers for other nodes
  - Help improve the stability of the network
- Standard nodes
  - Connect to supernodes and report list of files
  - Allows slower nodes to participate
- Search
  - Broadcast (Gnutella-style) search across supernodes
Topics Today

P2P Challenge 1: Efficient lookups
  Centralized
  Flooding
  BitTorrent
  Hashing-Based (Chord)

P2P Challenge 2: Fairness
  BitTorrent

P2P Challenge 3: Trust
  BitTorrent
  Blockchains
BitTorrent: Overview

• Swarming:
  • **Join**: contact centralized “tracker” server, get a list of peers.
  • **Publish**: Run a tracker server.
  • **Search**: Out-of-band. E.g., use Google to find a tracker for the file you want.
  • **Fetch**: Download chunks of the file from your peers. Upload chunks you have to them.

• Big differences from Napster:
  • Chunk based downloading
  • “few large files” focus
  • Anti-freeloading mechanisms
BitTorrent: Summary

• Pros:
  • Works reasonably well in practice
  • Efficient out of band search mechanism

• Cons:
  • Central tracker server needed to bootstrap swarm
    • Serverless tracker designs exist
  • Tracker needs to maintain peer list for every file
BitTorrent: Summary

• Pros:
  • Works reasonably well in practice
  • Gives peers incentive to share resources; avoids freeloaders

• Cons:
  • Pareto Efficiency relative weak condition
  • Central tracker server needed to bootstrap swarm
    • Alternate tracker designs exist (e.g. DHT based)
Topics Today

P2P Challenge 1: Efficient lookups
- Centralized
- Flooding
- BitTorrent
- Hashing-Based (Chord)

P2P Challenge 2: Fairness
- BitTorrent

P2P Challenge 3: Trust
- BitTorrent
- Blockchains
DHT: Overview (1)

- Goal: decentralized + quick search
  - item (file) identified is always found in a reasonable # of steps without a centralized server
- Abstraction: a distributed hash-table (DHT) data structure
  - insert(id, item);
  - item = query(id);
  - Note: item can be anything: a data object, document, file, pointer to a file…
- Implementation: nodes in system form a distributed data structure
  - Can be Ring, Tree, Hypercube, Skip List, Butterfly Network, ...
DHT: Overview (2)

- Structured Overlay Routing:
  - **Join**: On startup, contact a “bootstrap” node and integrate yourself into the distributed data structure; get a node id
  - **Publish**: Route publication for file id toward a close node id along the data structure
  - **Search**: Route a query for file id toward a close node id. Data structure guarantees that query will meet the publication.
  - **Fetch**: Two options:
    - Publication contains actual file → fetch from where query stops
    - Publication says “I have file X” → query tells you 128.2.1.3 has X, use IP routing to get X from 128.2.1.3
DHT: Consistent Hashing

A key is stored at its successor: node with next higher ID
Routing: Chord Basic Lookup

Where is K80?

Yes

Where is K80?

Yes

Where is K80?
Routing: Chord Basic Lookup

- Iterative routing
  - Similar to iterative DNS resolution
- Can also be recursive
  - Reduces some RTTs
- Still O(N), slow
Routing: Finger table - Faster Lookups
Routing: Chord

• Associate to each node and item a unique id in an uni-dimensional space

• Properties
  • Routing table size $O(\log(N))$, where $N$ is the total number of nodes
  • Guarantees that a file is found in $O(\log(N))$ steps
Routing: Chord Summary

- Assume identifier space is $0 \ldots 2^m$
- Each node maintains
  - Finger table
    - Entry $i$ in the finger table of $n$ is the first node that succeeds or equals $n + 2^i$
  - Predecessor node
- An item identified by $id$ is stored on the successor node of $id$
Routing: Chord Example

• Assume an identifier space 0..7

• Node n1:(1) joins→all entries in its finger table are initialized to itself
Routing: Chord Example

Succ. Table

<table>
<thead>
<tr>
<th>i</th>
<th>id+2i</th>
<th>succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
Routing: Chord Example

- Node n2:(2) joins
Routing: Chord Example

<table>
<thead>
<tr>
<th>i</th>
<th>id+2^i</th>
<th>succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>id+2^i</th>
<th>succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>
Routing: Chord Example

- Nodes n3:(0), n4:(6) join
Routing: Chord Example

<table>
<thead>
<tr>
<th>i</th>
<th>id+2^i</th>
<th>succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>id+2^i</th>
<th>succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>id+2^i</th>
<th>succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>
Routing: Chord Example

<table>
<thead>
<tr>
<th>i</th>
<th>id+2(^i)</th>
<th>succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>id+2(^i)</th>
<th>succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>id+2(^i)</th>
<th>succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>id+2(^i)</th>
<th>succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

K7

K2
Routing: Query

• Upon receiving a query for item \( id \), a node
• Check whether stores the item locally
• If not, forwards the query to the largest node in its successor table that does not exceed \( id \)
DHT: Discussion

• Pros:
  • Decentralized (except bootstrap node)
  • Guaranteed Lookup
  • $O(\log N)$ per node state and search scope

• Cons:
  • Supporting non-exact match search is hard
What can DHTs do for us?

• Distributed BitTorrent tracker
  • Use a DHT among peers instead of a centralized server for tracking
  • Still need a centralized bootstrap node for joining the DHT
  • Uses Kademlia DHT

• Distributed object lookup
  • Based on object ID
Topics Today

P2P Challenge 1: Efficient lookups
  Centralized
  Flooding
  BitTorrent
  Hashing-Based (Chord)

P2P Challenge 2: Fairness
  BitTorrent

P2P Challenge 3: Trust
  BitTorrent
  Blockchains
Fairness in P2P File Sharing Systems

- Freeloader problem
  - All peers must contribute
  - How to prevent peers leeching resources>
- Efficiency
  - Single peer is bottleneck => similar to bottleneck in a centralized approach
  - Load balancing across peers
BitTorrent: Sharing Strategy

- Employ “Tit-for-tat” sharing strategy
  - A is downloading from some other people
    - A will let the fastest N of those download from him
  - Be optimistic: occasionally let freeloaders download
    - Otherwise no one would ever start!
    - Also allows you to discover better peers to download from when they reciprocate
- Distribute chunks across peers
  - When a new file is seeded, peers download random chunks
  - Results in load balancing as different peers have different subset of chunks
Topics Today

P2P Challenge 1: Efficient lookups
   Centralized
   Flooding
   BitTorrent
   Hashing-Based (Chord)

P2P Challenge 2: Fairness
   BitTorrent

P2P Challenge 3: Trust
   BitTorrent
   Blockchains
P2P: Trust

- Cannot trust peers to do the right thing
  - Can provide malicious content
  - Can deliberately thwart attempts to access network
- Two aspects of trust
  - Is the data I get correct?
  - Are the peers behaving the way they should?
Topics Today

P2P Challenge 1: Efficient lookups
  Centralized
  Flooding
  BitTorrent
  Hashing-Based (Chord)

P2P Challenge 2: Fairness
  BitTorrent

P2P Challenge 3: Trust
  BitTorrent
  Blockchains
Bittorrent: Trust

• Check that the file is correct
  • Source of .torrent file provides this by allowing comments, upvotes and reporting of .torrent files

• Check that the chunk downloaded from a peer is valid
  • .torrent file has cryptographic hashes of each chunk, which is easy to verify but hard to thwart (SHA1)
  • Download from different peer if SHA1 fails for chunk, blacklist peer
Topics Today

P2P Challenge 1: Efficient lookups
- Centralized
- Flooding
- BitTorrent
- Hashing-Based (Chord)

P2P Challenge 2: Fairness
- BitTorrent

P2P Challenge 3: Trust
- BitTorrent
- Blockchains
Motivation: Decentralized Transactions

• Traditional transactions
  • Via trusted entities like banks/mints/lawyers
  • Transactions sometimes need to be reversed
    • Disputes, stolen credit cards
    • Requires mediation and additional trust (merchant → customer)
  • Significant transaction costs
    • Prevent emerging use cases, e.g., micro payments

• Decentralized transactions
  • Are they possible? Can you think of some challenges?
Challenges of Decentralized Transactions

• Key problem: double spending
  • Daniel has $5 - buys a $5-drink and a $5-sandwich at the same time

• Someone needs to keep track of ALL transactions
  • Traditional currency: mints
  • Internet: P2P distributed data-structure
Solution: Secure, Distributed Ledgers

• Ledger: every transaction ever made
• All participants need a copy

Steps to Maintain Distributed Ledger:

• 1) broadcast new transactions
• 2) each member collects transactions into a block
• 3) once block is full, broadcast, and move on

What if there’s a block collision?
Adding Consensus to Distributed Ledgers

Steps to Maintain Distributed Ledger:
• 1) broadcast new transactions
• 2) each member collects transactions into a block
• 3) reach consensus on next block
• 4) continue with 1)

What if someone has many IP addresses?
Solution: Blockchain Consensus

• 1) broadcast new transactions
• 2) each member collects transactions into a block
• 3) each member seeks proof-of-work for its block
  • proof-of-work (PoW): solve a computationally hard problem
• 4) member who finds PoW broadcasts block+PoW
• 5) other member check block, seek next PoW
• 6) consensus over time
Blockchain in More Detail

- Blocks contain transactions
- Chain of blocks secured using cryptographic hashes
- Each block contains cryptographic hash of previous block
- Tampered block can easily be checked for
Blockchain Proof-of-Work

- Idea: one vote per CPU
- Hashcash cryptographic puzzle used in Bitcoin
- Find nonce such that SHA-256 hash of (block + nonce) has K leading zeros

```
prev_hash: (...)  
Txn 1  
Txn 2  
Txn 3  
Txn 4  
.  
.  
Txn N  
Random value (nonce)
```

SHA-256 hash

hash < target ?
Achieving consensus (2)

prev_hash: (...)

Txn 1
Txn 2
Txn 3
Txn 4
.
.
.
Txn N

Random value: (guess)

SHA-256 hash

hash < target?
How Do Member Verify Blocks?

- Many transactions are included in a single block
- Single transaction may need to be verified
  - Naive: Scan the entire block - slow
  - Optimization: Merkle trees

![Merkle Tree Diagram]

https://bitcoin.stackexchange.com/a/50680
Merkle trees

• Leaf nodes are data blocks
• Non-leaf nodes are cryptographic hashes of child nodes
• Verify data contained in sub-tree using hash of sub-tree root
• Efficiently verify single data block using a few hash values
Merkle trees

https://bitcoin.stackexchange.com/a/50680
Security Guarantee of a Blockchain

- To modify old transactions, proof of work has to be redone for all successive committed blocks
- 51 attack
  - If an organization has more than 51% of the total compute, it can choose which transactions get committed
  - Very hard to change older blocks even with a majority of computational power
Incentivizing proof of work (mining)

- Mining is the process of generating proof of work
- Miner adds reward to self at the beginning of the block
- If the miner’s block gets added to the blockchain, miner receives a reward
Example: bitcoin

Ledger: transactions of bitcoin currency payments
Reward: bitcoins

Introduced in 2009 by “Satoshi Nakamoto”
(not known publicly)
In use today
(10 million transactions / month)
Example: Namecoin

Ledger: Names and IP addresses of various servers, along with namecoin transactions

Reward: Namecoins, which are just like bitcoins

Introduced in 2011

Censor-free fully p2p naming system

“decentralized DNS”
P2P: Summary

- Peer-to-Peer designs can be very scalable
- Three issues: Lookup, fairness and trust
- Lessons learned:
  - Many kinds of lookup: Centralized, flooding, structured
  - Still need bootstrapping nodes, true decentralization is hard
  - Needs incentives and other mechanisms to guarantee fairness and participation
  - Trust can be achieved via cryptographic techniques and blockchains
P2P: Summary

• Many different styles; remember pros and cons of each
  • centralized, flooding, unstructured and structured routing
• Lessons learned:
  • Flooding messages to everyone is bad
  • Need incentives to discourage freeloading and encourage participation
  • Security can be achieved through proof of work
BitTorrent: Publish/Join

Tracker
BitTorrent: Fetch
When are p2p / DHTs useful?

• Caching and “soft-state” data
  • Works well! BitTorrent, KaZaA, etc., all use peers as caches for hot data

• Finding read-only data
  • Limited flooding finds hay
  • DHTs find needles

• BUT…
A Peer-to-peer Google?

- Complex intersection queries ("the" + "who")
  - Billions of hits for each term alone
- Sophisticated ranking
  - Must compare many results before returning a subset to user
- Very, very hard for a DHT / p2p system
  - Need high inter-node bandwidth
  - (This is exactly what Google does - massive clusters)
Writable, persistent p2p

• Do you trust your data to 100,000 monkeys?
• Node availability hurts
  • Ex: Store 5 copies of data on different nodes
  • When someone goes away, you must replicate the data they held
  • Hard drives are *huge*, but home upload bandwidth is tiny - perhaps 50 Gbytes/day
  • Takes many days to upload contents of 1TB hard drive. Very expensive leave/replication situation!