LAU
الجامعة اللبنانية الأميركية
Lebanese American University

CSC 634: Networks Programming

Lecture 09: Peer to Peer Computing (Continued)

Instructor: Haidar M. Harmanani

Recap: Classification of P2P Systems

- 1st generation
  - Typically: Napster
- 2nd generation
  - Typically: Gnutella
- 3rd generation
  - Typically: Superpeer networks
- 4th generation
  - Typically: Distributed hash tables
  - Note: For DHTs, no division into generations yet

Trackerless Torrents

- New versions of BitTorrent have the ability to locate swarms without a tracker
  - Based on a P2P overlay
    - A network that is constructed by the Internet peers in the application layer on top of the IP network.
  - Distributed hash table (DHT)
- Recall: peers located via DHT are given “H” state

Distributed Hash Tables: Motivation

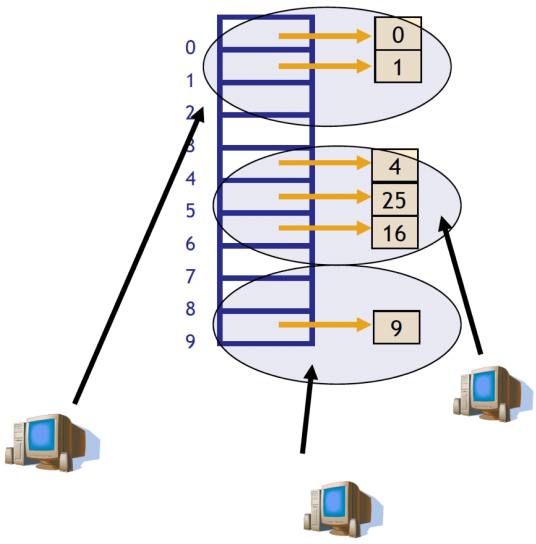
- Searching in P2P networks is not efficient
  - Either centralized system with all its problems ***or*** distributed system with all its problems
  - Hybrid systems cannot guarantee discovery either
- Actual file transfer process in P2P network is scalable
  - File transfers directly between peers
- Searching does not scale in same way
- Original motivation for DHTs: more efficient searching and object location in P2P networks
- Put another way: Use addressing instead of searching

Problems Solved by DHT

- DHT solves the problem of mapping keys to values in the distributed hash table
- Efficient storage and retrieval of values
- Efficient routing
  - Robust against many failures
  - Efficient in terms of network usage
- Provides hash table-like abstraction to application

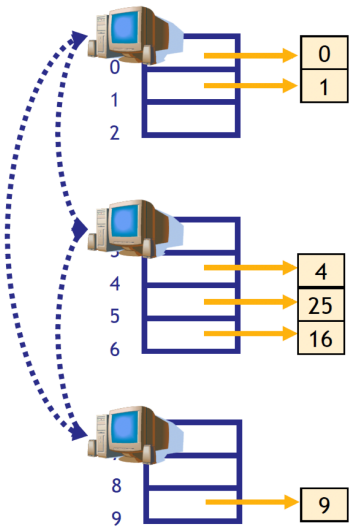
Distributed Hash Table: Idea

- Hash tables are fast for lookups
- Idea: Distribute hash buckets to peers
- Result is Distributed Hash Table (DHT)
- Need efficient mechanism for finding which peer is responsible for which bucket and routing between them



DHT: Principle

- In a DHT, each node is responsible for one or more hash buckets
  - As nodes join and leave, the responsibilities change
- Nodes communicate among themselves to find the responsible node
  - Scalable communications make DHTs efficient
- DHTs support all the normal hash table operations
- Hash buckets distributed over nodes
- Nodes form an overlay network
  - Route messages in overlay to find responsible node
- Routing scheme in the overlay network is the difference between different DHTs
- DHT behavior and usage:
  - Node knows "object" name and wants to find it
    - Unique and known object names assumed
  - Node routes a message in overlay to the responsible node
  - Responsible node replies with "object"
    - Semantics of "object" are application defined



DHT Examples

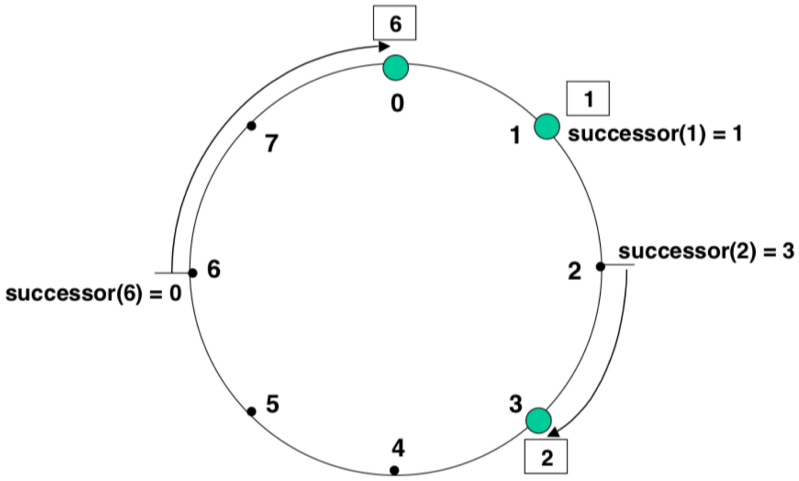
- In the following we look at Chord, an example DHTs
- Several others exist too
  - CAN, Tapestry
  - Pastry, Plaxton, Kademlia, Koorde, Symphony, P-Grid, CARP, …
- All DHTs provide the same abstraction:
  - DHT stores key-value pairs
  - When given a key, DHT can retrieve/store the value
  - No semantics associated with key or value
- Difference is in overlay routing scheme

Chord

- Developed at MIT
- Originally published in 2001 at Sigcomm conference
- Chord's overlay routing principle quite easy to understand
  - Paper has mathematical proofs of correctness and performance
- Many projects at MIT around Chord
  - CFS storage system
  - Ivy storage system
  - Plus many others…

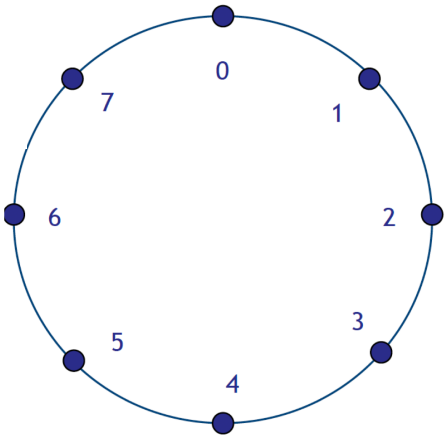
Chord: Basics

- Chord uses SHA-1 hash function
  - Results in a 160-bit object/node identifier
  - Same hash function for objects and nodes
- Node ID hashed from IP address
- Object ID hashed from object name
  - Object names somehow assumed to be known by everyone
- SHA-1 gives a 160-bit identifier space
- Organized in a ring which wraps around
  - Nodes keep track of predecessor and successor
  - Node responsible for objects between its predecessor and itself
  - Overlay is often called "Chord ring" or "Chord circle"



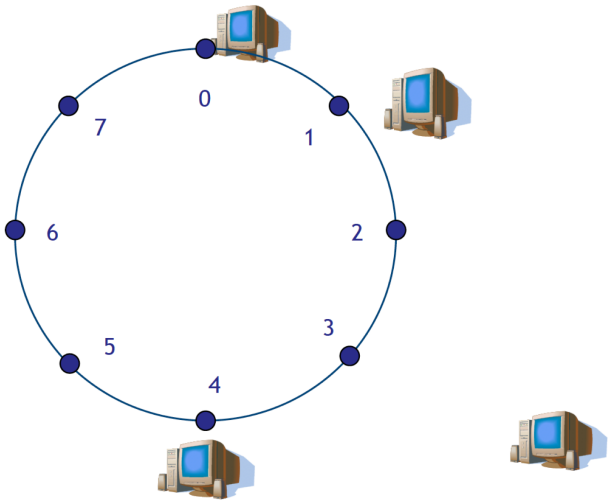
Chord Joining: Example

- Setup: Existing network with nodes on 0, 1 and 4
- Note: Protocol messages simply examples
- Many different ways to implement Chord
  - Here only conceptual example
  - Covers all important aspects

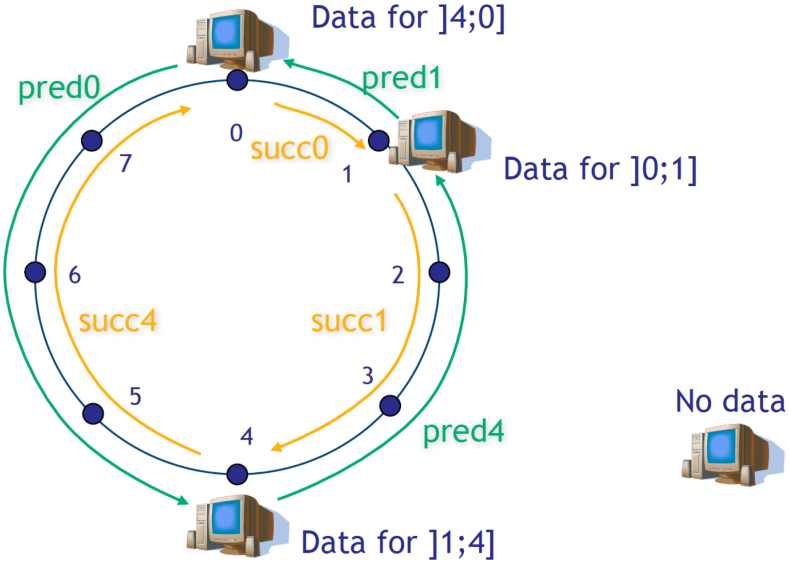


Chord Joining: Example – Start

- New node wants to join
- Hash of the new node: 6
- Known node in network: Node 1
- Contact Node 1
  - Include own hash



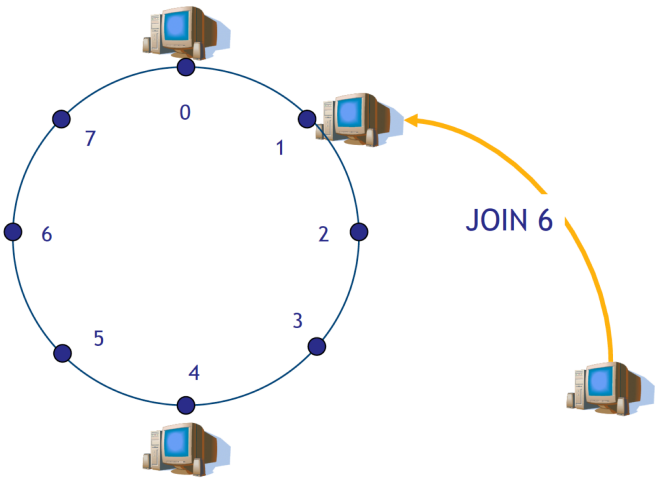
Chord Joining: Example – Situation Before Join



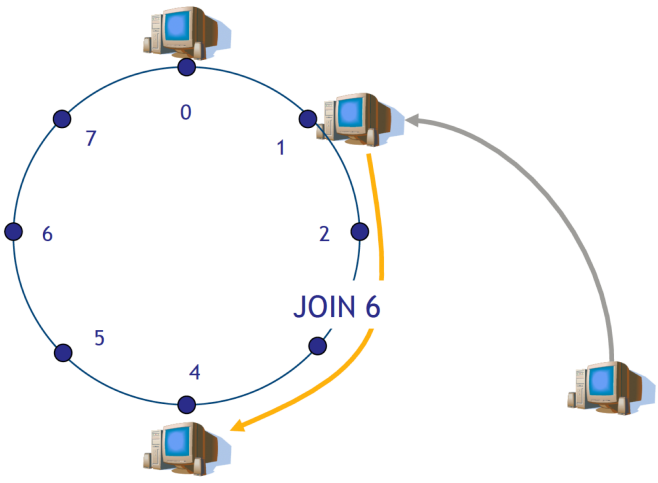
Chord Joining: Example – Join gets routed along the network

- New node wants to join
- Hash of the new node: 6
- Known node in network: Node 1
- Contact Node 1
  - Include own hash

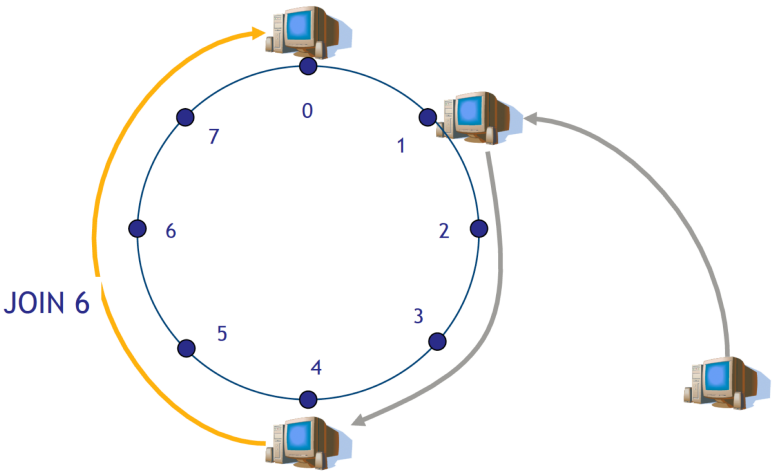
- Arrows indicate open connections
- Example assumes connections are kept open, i.e., messages processed recursively
- Iterative processing is also possible



Chord Joining: Example – Successor of New Node Found

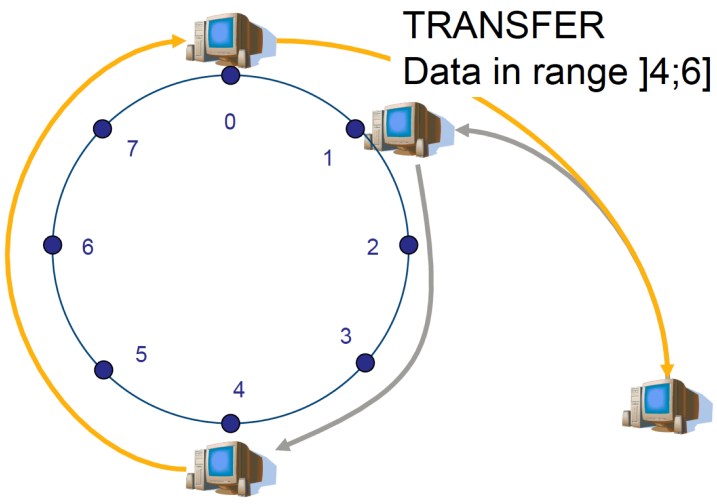


Chord Joining: Example – Join gets routed along the network

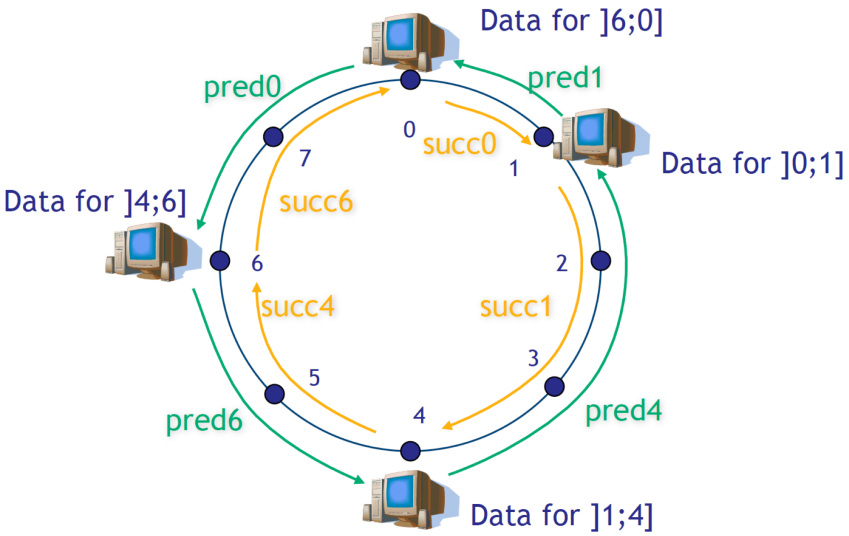


Chord Joining: Example – Joining Successful + Transfer

- Joining is successful
- Old responsible node transfers data that should be in new node
- New node informs Node4 about new successor (not shown)
- Note: Transferring can happen also later

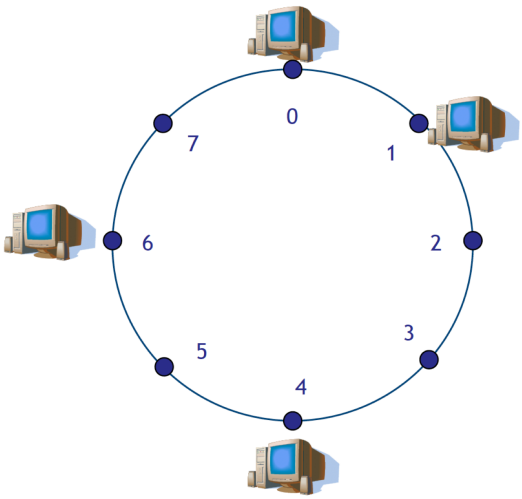


Chord Joining: Example – All Is Done

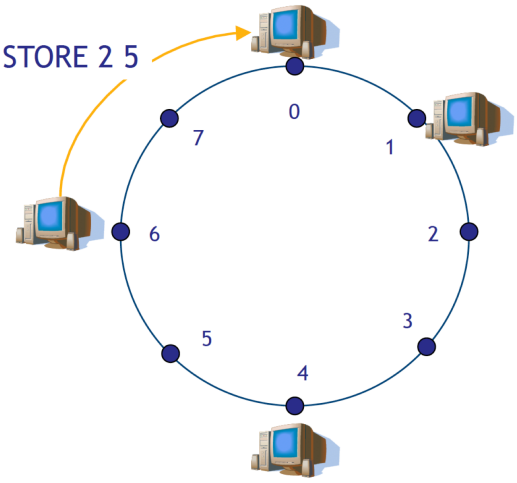


Storing a Value

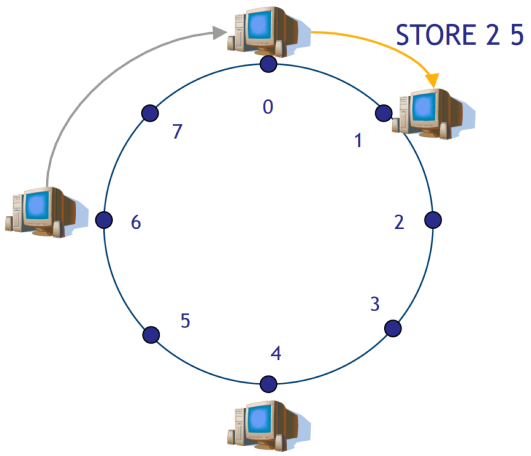
- Node 6 wants to store object with name "Foo" and value 5
- hash(Foo) = 2
- Data are stored at the successor
  - Identifiers are represented as a circle of numbers from 0 to 2^m
  - Key K is assigned the first node whose key is equal or follows k in the identifier list



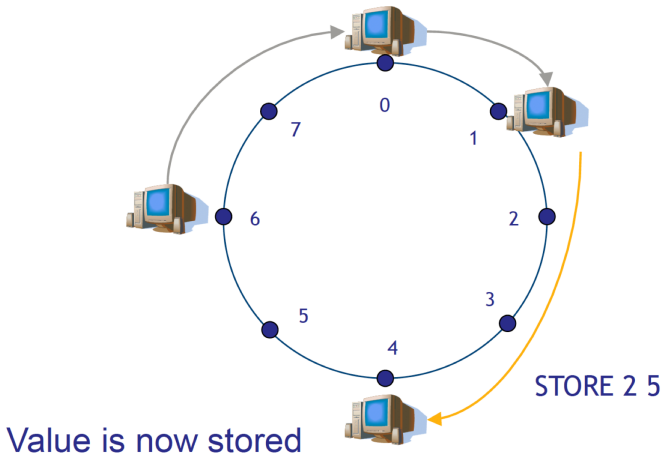
Storing a Value



Storing a Value

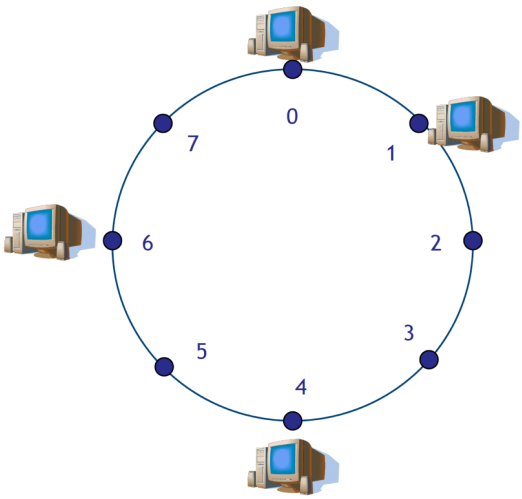


Storing a Value

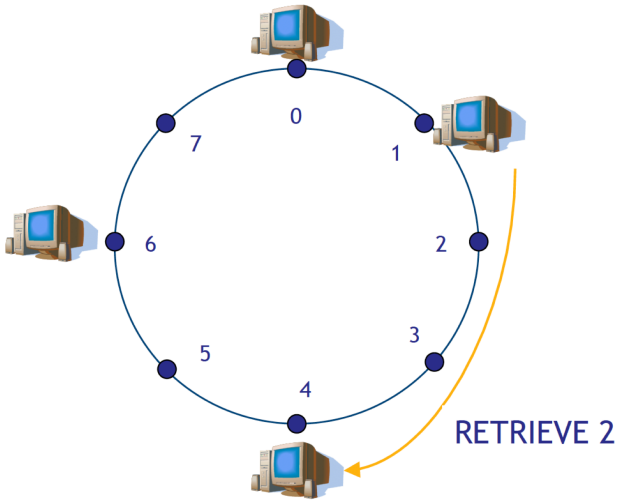


Retrieving a Value

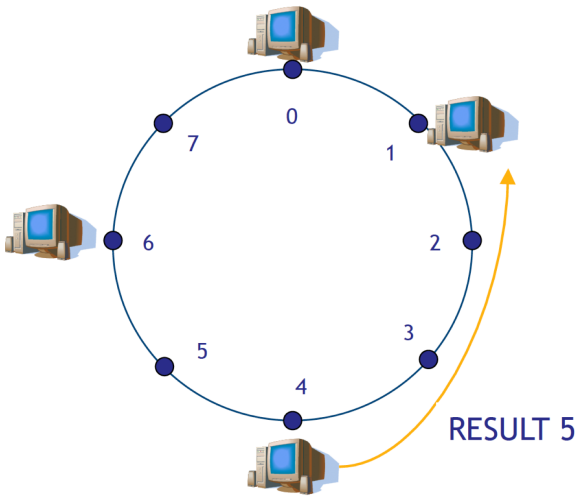
- Node 1 wants to get object with name “Foo”
- hash(Foo) = 2 ▶ Foo is stored on node 4



Retrieving a Value



Retrieving a Value



Chord: Scalable Routing

- Routing happens by passing message to successor
- What happens when there are 1 million nodes?
  - On average, need to route 1/2-way across the ring
  - In other words, 0.5 million hops! Complexity O(n)
- How to make routing scalable?
  - Answer: Finger tables
- Basic Chord keeps track of predecessor and successor
- Finger tables keep track of more nodes
  - Allow for faster routing by jumping long way across the ring
  - Routing scales well, but need more state information
- Finger tables not needed for correctness, only performance improvement

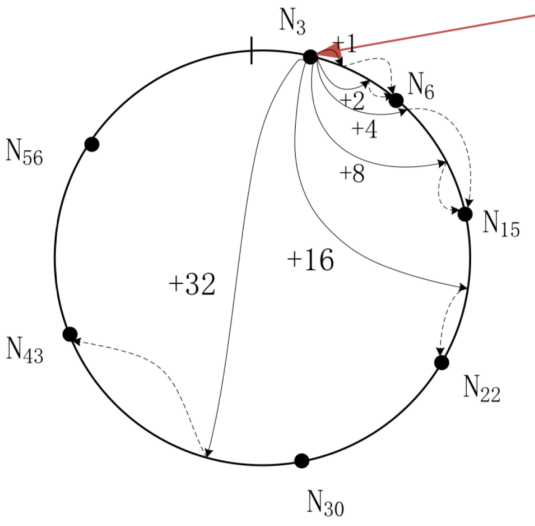
Chord: Finger Tables

- In *m*-bit identifier space, node has up to *m* fingers
- Fingers are stored in the finger table
- Row *i* in finger table at node n contains first node s that succeeds n by at least 2^{i-1} on the ring
- In other words:
  - finger[i] = successor(n + 2^{i-1})
- First finger is the successor
- Distance to finger[i] is at least $^{2i-1}$

Chord: Scalable Routing

- Finger intervals increase with distance from node n
  - If close, short hops and if far, long hops
- Two key properties:
  - Each node only stores information about a small number of nodes
  - Cannot in general determine the successor of an arbitrary ID
  - Example has three nodes at 0, 1, and 4
  - 3-bit ID space --> 3 rows of fingers

An example of finger table



finger table	
N_3+1	N_6
N_3+2	N_6
N_3+4	N_{15}
N_3+8	N_{15}
N_3+16	N_{22}
N_3+32	N_{43}

H. Zhang, Y. Wen, H. Xie, and N. Yu, "A Survey on Distributed Hash Table (DHT): Theory, Platforms, and Applications," 2013.

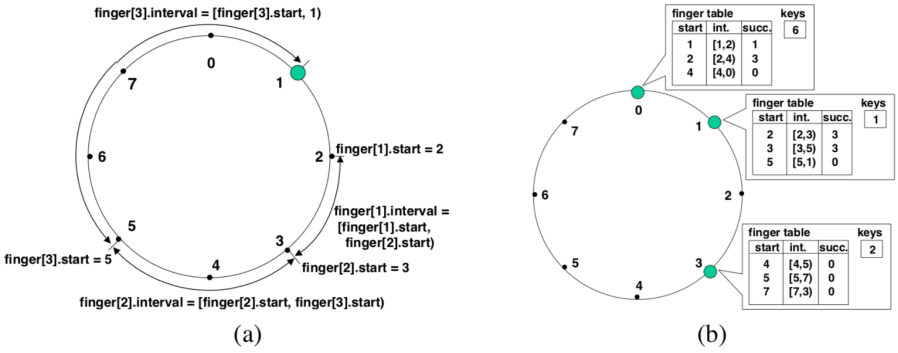


Figure 3: (a) The finger intervals associated with node 1. (b) Finger tables and key locations for a net with nodes 0, 1, and 3, and keys 1, 2, and 6.

Chord: Performance

- Search performance of "pure" Chord O(n)
  - Number of nodes is n
- With finger tables, need O(log n) hops to find the correct node
  - Fingers separated by at least 2^{i-1}
  - With high probability, distance to target halves at each step
  - In beginning, distance is at most 2m
  - Hence, we need at most m hops
- For state information, "pure" Chord has only successor and predecessor, O(1) state
- For finger tables, need m entries
  - Actually, only O(log n) are distinct
  - Proof is in the paper

BitTorrent and TCP

- BitTorrent accounts for 35-70% of all Internet traffic
- Thus, BitTorrent's behavior impacts everyone
- BitTorrent's use of TCP causes problems
  - Long lived, BitTorrent TCP flows are "elephants"
    - Ramp up past slow start, dominate router queues
  - Many applications are "mice," get trampled by elephants
    - Short lived flows (e.g. HTTP traffic)
    - Delay sensitive apps (i.e. VoIP, SSH, online games)
- Have you ever tried using SSH while using BitTorrent?

Making BitTorrent Play Nice

- Key issue: long-lived TCP flows are aggressive
  - TCP is constantly probing for more bandwidth
  - TCP induces queuing delay in the network
- Does BitTorrent really need to be so aggressive?
  - BitTorrent is not delay sensitive
    - Do you care if your download takes a few minutes longer?
  - BitTorrent is low-priority background traffic
    - You probably want to do other things on the Internet while BitTorrent is downloading
- Solution: use less a less aggressive transport protocol for BitTorrent

Micro Transport Protocol (µTP)

- Designed by BitTorrent, Inc.
- UDP-based transport protocol
- Uses LEDBAT principals
  - Low Extra Delay Background Transport
  - Used by Apple for software updates
- Duplicates many TCP features
  - Window based sending, advertised windows
  - Sequence numbers (packet based, not byte based)
  - Reliable, in-order packet delivery
- Today: widely adopted by BitTorrent clients and open-sourced

µTP and LEDBAT

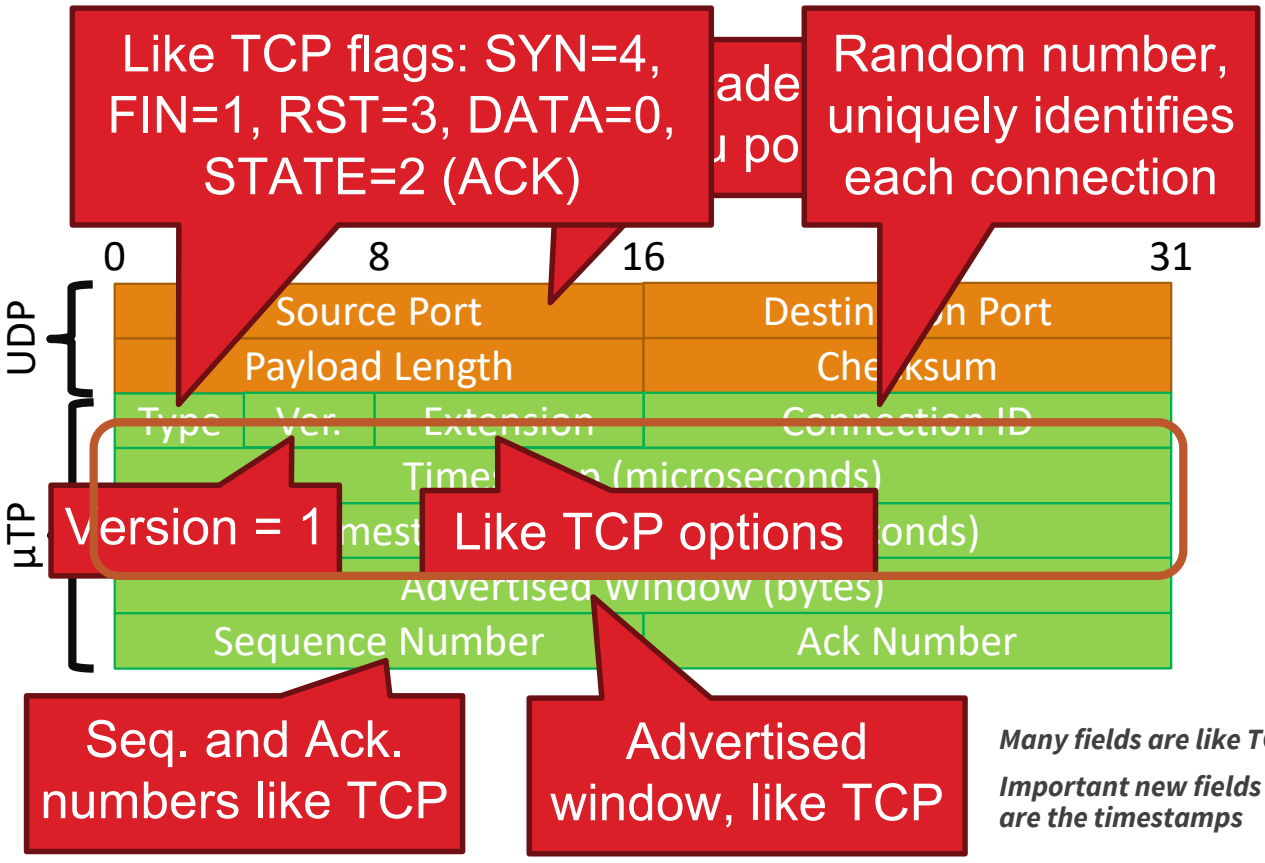
- µTP is based on IETF LEDBAT standard (RFC 6817)
- Low Extra Delay Background Transport
  - Low delay congestion control algorithm
- Goal: fast transfer of bulk data in the background
  - Use all available bandwidth (fast transfer speed)
  - … but, do not starve other applications
    - Background data transfer is not delay sensitive
    - Backoff gracefully and give bandwidth to delay sensitive applications
    - Does not increase queuing delay on the path

LEDBAT Details

- Delay-based congestion control protocol
  - Similar algorithm to TCP Vegas
  - Measure one-way delay, reduce rate when delay increases
- Constraint: be less aggressive than TCP
  - React early to congestion and slow down
  - Do not induce queuing delay in the network
- LEDBAT is a “scavenger” cc protocol
  - Scavenge unused bandwidth for file transfer
  - … but don’t take bandwidth from other flows

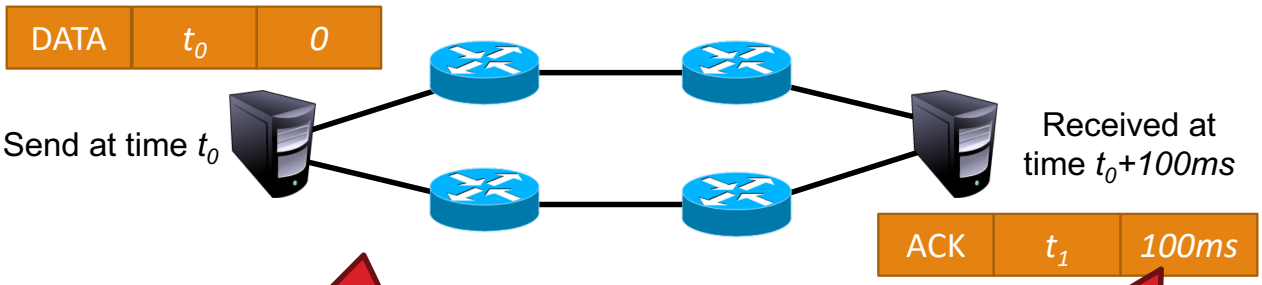
LEDBAT Details

- Many fields are like TCP
- Important new fields are the timestamps



Timestamps and Delay

- Timestamps used to measure one-way delay
  - Timestamp: time at which packet was sent
  - Timestamp Difference: sent time – received time



Sender knows one-way delay = 100ms

Time difference inserted into ACK

µTP tries to keep one-way delay ~100ms

Estimate the baseline delay on the path

- 2 minutes])

Is delay below our target (positive value), or above our target (negative value)

Current delay on the path above the

Time difference fr most recent AC

Convert units from "time" to "packets"

Finally, adjust the window size (may be + or – adjustment)

- scaled * window_factor
- max_window = max_window + scaled_gain

More µTP Details

- Delay-based mechanism replaces slow start and additive increase
- What if a packet drops?
  - max_window = max_window * 0.5 (just like TCP)
- What if off_target is a large negative number?
  - max_window = 1 packet (don't starve the connection)
- Error handling in µTP :
  - Uses RTO like Tahoe to retransmit lost packets
  - Uses fast retransmit like TCP Reno

Discussion

- In this case, developing a new transport protocol was (arguably) the right decision
  - BitTorrent generates huge amounts of traffic
  - Whole Internet benefits if BitTorrent is more friendly
- However, inventing new protocols is hard
  - µTP reimplements most of TCP
    - RTO estimation, Nagle's algorithm, etc.
  - Early version of µTP performed much worse than TCP
    - Lots of bugs related to packet pacing and sizing
- Takeaway: develop new transport protocols only if absolutely necessary

Problems NOT Solved by DHT

- Robustness
  - No guarantees against big failures
  - Threat models for DHTs not well-understood yet
- Availability
  - Data not guaranteed to be available
  - Only probabilistic guarantees (but possible to get high prob.)
- Consistency
  - No support for consistency
  - Data in DHT often highly replicated, consistency is a problem
- Version management
  - No support for version management
  - Might be possible to support this to some degree

P2P File Systems

- P2P filesystems (FS) or P2P storage systems were the first applications of DHTs
- Fundamental principle:
  - Key = filename, Value = file contents
- Different kinds of systems
  - Storage for read-only objects
  - Read-write support
  - Stand-alone storage systems
  - Systems with links to standard filesystems (e.g., NFS)

Motivation of P2P File Systems

- Why build P2P filesystems?
- Light-weight, reliable, wide-area storage
  - At least in principle…
- Distributed filesystems not widely deployed either…
  - Were studied already long time ago
- Gain experience with DHT and how DHTs could be used in real applications
  - DHT abstraction is powerful, but it has limitations
  - Understanding of the limitations is valuable

P2P FS: Basic Technique

- Three fundamental basic techniques for building distributed storage systems
  - Splitting files into blocks
  - Replicating files (or blocks!)
  - Using logs to allow modifications
- For now: Simple analysis of advantages and disadvantages and some examples

Splitting Files into Blocks

- Motivation
  - Files are of different sizes and peers storing large files have to serve more data
- Pro:
  - Dividing files into equal-sized blocks and storing blocks on different peers can achieve load balance
  - If different files share blocks, we can save on storage
- Con:
  - Instead of needing one peer online, all peers with all blocks must be online (see below)
  - Need metadata about blocks to be stored somewhere
  - Granularity tradeoff: Small blocks -> Good load balance, but lots of overhead and vice versa

Replication

- Motivation
  - If file (or block) is stored only on one peer and that peer is offline, data is not available
- Replicating content to multiple peers significantly increases content availability
- Pro:
  - High availability and reliability
    - But only probabilistic guarantees
- Con:
  - How to coordinate lots of replicas?
    - Especially important if content can change
  - Unreliable network requires high degree of replication for decent availability
    - "Wastes" storage space

Logs

- Motivation
  - If we want to change the stored files, we need to modify every stored replica
- Keep a log for every file (user, …) which gives information about the latest version
- Pro:
  - Changes concentrated in one place
  - Anyone can figure out what is the latest version
- Con:
  - How to keep the log available?
  - By replicating it? ;-)

P2P FS: Examples

- CFS (blocks and replication)
  - Basic, read-only system
  - Based on Chord
- OceanStore (replication)
  - Vision for a global storage system
  - Based on Tapestry
- Ivy (logs)
  - Read-write, provide NFS semantics
  - Based on Chord

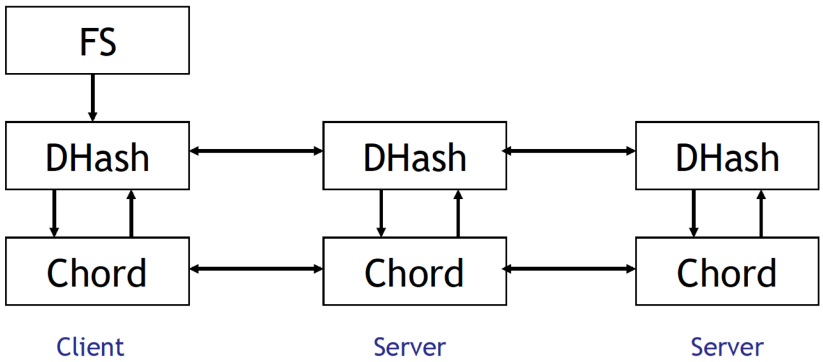
Cooperative File System (CFS)

- Developed at MIT, by same people as Chord
- CFS based on the Chord DHT
- Read-only system, only 1 publisher
- CFS stores blocks instead of whole files
  - Part of CFS is a generic block storage layer
- Features:
  - Load balancing
  - Performance equal to FTP
  - Efficiency and fast recovery from failures

CFS Properties

- Decentralized control
- Scalability (comes from Chord)
- Availability
  - In absence of catastrophic failures, of course…
- Load balance
  - Load balanced according to peers' capabilities
- Persistence
  - Data will be stored for as long as agreed
- Quotas
  - Possibility of per-user quotas
- Efficiency
  - As fast as common FTP in wide area

CFS: Layers, Clients, and Servers



- FS: provide filesystem API, interpret blocks as files
- DHash: Provides block storage
- Chord: DHT layer, slightly modified
- Clients access files, servers just provide storage

Chord Layer in CFS

- Chord layer is slightly modified from basic Chord
- Instead of 1 successor, each node keeps track of r successors
- Finger tables as before
- Also, try to reduce lookup latency
  - Nodes measure latencies to other nodes
  - Report measured latencies to other nodes

DHash Layer

- DHash stores blocks as opposed to whole files
  - Better load balancing
  - More network query traffic, but not really significant
- Each block replicated k times, with $r \geq k$
- Two kinds of blocks:
  - Content block, addressed by hash of contents
  - Signed blocks (= root blocks), addressed by public key
    - Signed block is the root of the filesystem
    - One filesystem per publisher
- Blocks are cached in network
  - Most of caching near the responsible node
  - Blocks in local cache replaced according to least-recently used
  - Consistency not a problem, blocks addressed by content
    - Root blocks different, may get old (but consistent) data

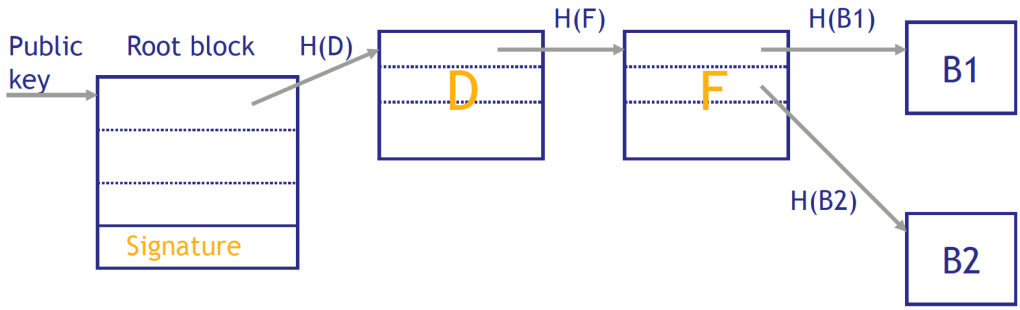
DHash API

- DHash provides following API to clients:

Put_h(block)	Store block
Put_s(block, pubkey)	Store or update signed block
Get(key)	Fetch block associated with key

- Filesystem starts with root (= signed) block
- Block under publisher's public key and signed with private key
- For clients, read-only, publisher can modify filesystem by inserting a new root block
- Root block has pointers to other blocks
  - Either piece of a file or filesystem metadata (e.g., directory)
- Data stored for an agreed-upon finite interval

CFS Filesystem Structure



- Root block identified by publisher's public key
  - Each publisher has its own filesystem
  - Different publishers are on separate filesystems
- Other blocks identified based on hash of contents
- Other blocks can be metadata or pieces of file

Load Balancing and Quotas

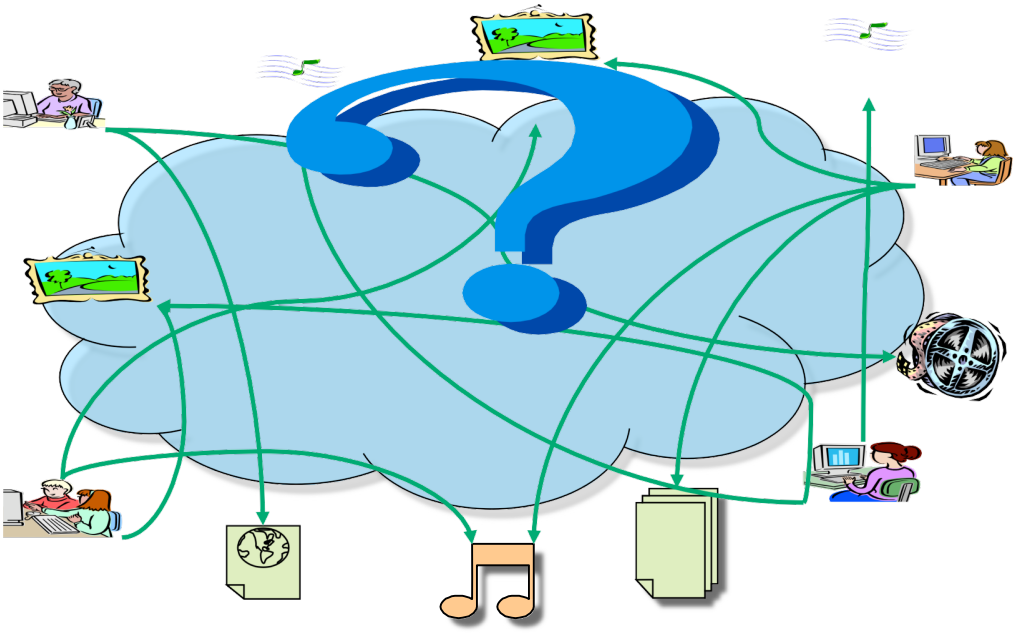
- Different servers have different capabilities
- One real server can run several virtual servers
- Number of virtual servers depends on the capabilities
  - "Big" servers run more virtual servers
- CFS operates at virtual server level
  - Virtual nodes on a single real node know each other
  - Possible to use short cuts in routing
- Quotas can be used to limit storage for each node
- Quotas set on a per-IP address basis
  - Better quotas require central administration
  - Some systems implement "better" quotas, e.g., PAST

Updates in CFS

- Only publisher can change data in filesystem
- CFS will store any block under its content hash
  - Highly unlikely to find two blocks with same SHA-1 hash
  - No explicit protection for content blocks needed
- Root block is signed by publisher
  - Publisher must keep private key secret
- No explicit delete operation
  - Data stored only for agreed-upon period
  - Publisher must refresh periodically if persistence is needed

Content Distribution

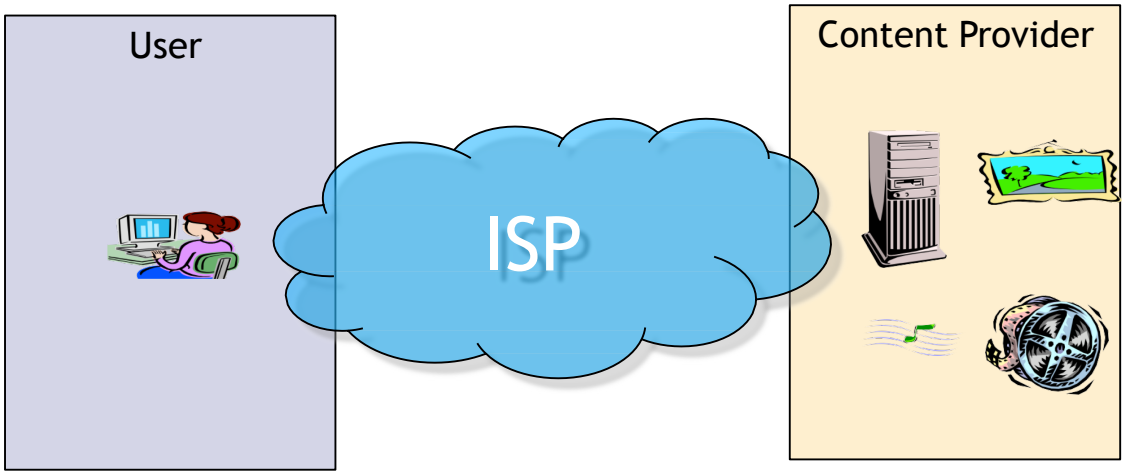
What is the Content Distribution Problem?



Problem Definition

- How to distribute (popular) content efficiently on Internet to a large number of geographically distributed people?
- …while keeping your costs manageable! :-)
- Who is "you"?
- So, we need to answer the following questions…
  - Who are the players?
  - What is the content?
  - How are costs determined?
  - How is performance measured?

The Players



- ISP represents the network path between the user and the content provider
- Typically, such a path contains several ISPs

Roles of the Players

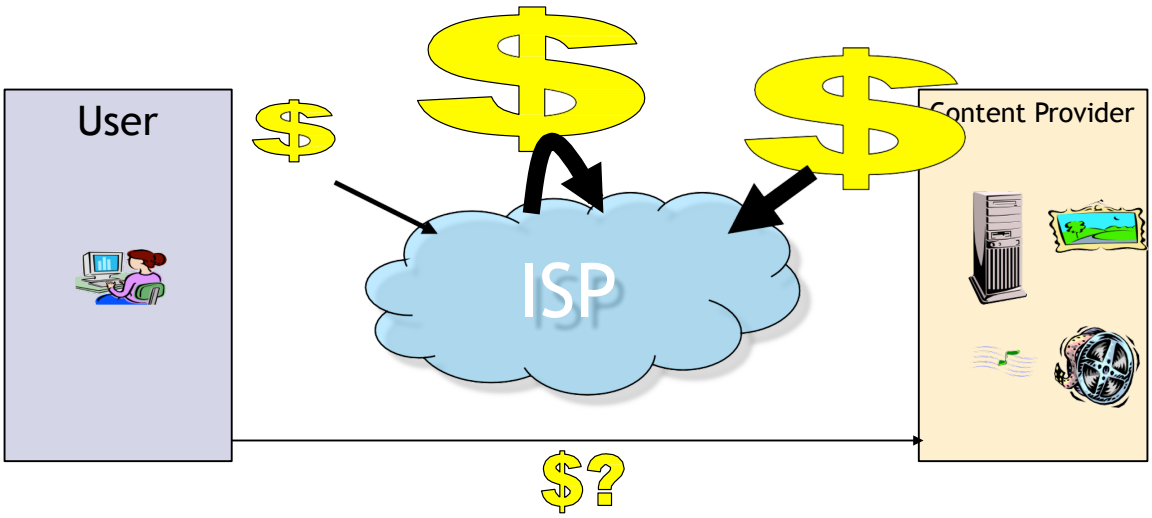
- User
  - Wants to access content provided by content provider
  - Buys access to network from ISP
  - Possibly pays content provider for content
- Content provider
  - Produces content
  - Runs a server (or outsources this)
  - Buys access from an ISP
- ISP
  - Provides the network connection for the others

What Is Content?

- Content is any digital content the users want
- Examples:
  - Web pages
  - Music files
  - Videos
  - Streaming media
  - and so on…

Follow the Money

- Where does the money come from?



Business Relationships

- User
  - Pays money to ISP for connectivity
  - These days often flat-rate, can be per-byte
  - Might rarely pay content provider
- Content provider
  - Pays ISP for connectivity
  - Pays for running a server
  - Needs to get money from somewhere
  - May or may not need to make a profit from content
- ISP
  - Gets money from others
  - Uses money to run network
  - Wants to make a profit

Performance Targets

- User
  - Wants content fast
  - Does not want to pay (too much)
- Content provider
  - Wants to get as many users as possible
  - While maintaining costs as low as possible
- ISP
  - Wants to make as much money as possible
- Which goal is the most important?

- In real world, user needs often not "relevant"
  - Users do not contribute enough money
  - Users might not have a choice, e.g., only 1 ISP possible
- ISP is the most important in many cases
- Content providers have some say in some cases
  - Compare proxy caching and CDNs

Content Distribution Technology

- Content distribution is an "old" problem
  - Read: Originates from the Web

- Technology currently in use:
  - Client-side web proxies/caches
  - Server-side load balancing
  - Mirror servers
  - Content Distribution Networks (CDN)
  - Peer-to-peer content distribution
  - Multicast (at least in theory…)

File Sharing?!?

- Is file sharing content distribution?
- YES:
  - There is content being distributed
- NO:
  - Goal is to make content available, not the actual distribution
  - Main focus is on searching and locating content
  - Actual distribution just a simple download with nothing fancy
- We exclude file sharing from content distribution

Peer-to-Peer Content Distribution

- Definition
  - Peer-to-peer content distribution is content distribution which is implemented in a peer-to-peer fashion
- Best/only current example: BitTorrent
  - One file, replicate to everyone as fast as possible
- Why P2P content distribution?

P2P Content Distribution: Pros and Cons

- Disadvantages
  - Users must want to help
    - Incentives?
  - Need to trust users
  - No (easy) way to optimize
    - Topological locality
  - P2P has “bad reputation”
- Advantages
  - Cheap for content provider
    - Need to seed file, that’s it!
    - Server farms, CDNs, etc. are very, very expensive
  - Capacity increases with increasing number of users

Current State

- Real world
  - BitTorrent and nothing much else
- Research world:
  - BitTorrent under active research
    - Piece and peer selection algorithms
    - Measurement work
  - Avalanche from Microsoft
    - Same as BitTorrent, but uses network coding
  - P2P Streaming
    - Lots of proposals

Network Coding: Motivation

- Main question in BitTorrent: Which piece to download next?
- BitTorrent's answer: Rarest first!
- Rare blocks might disappear from system?
- How about advanced coding techniques?
  - Erasure codes at the source
  - Network coding by the peers

Erasure Codes at the Source

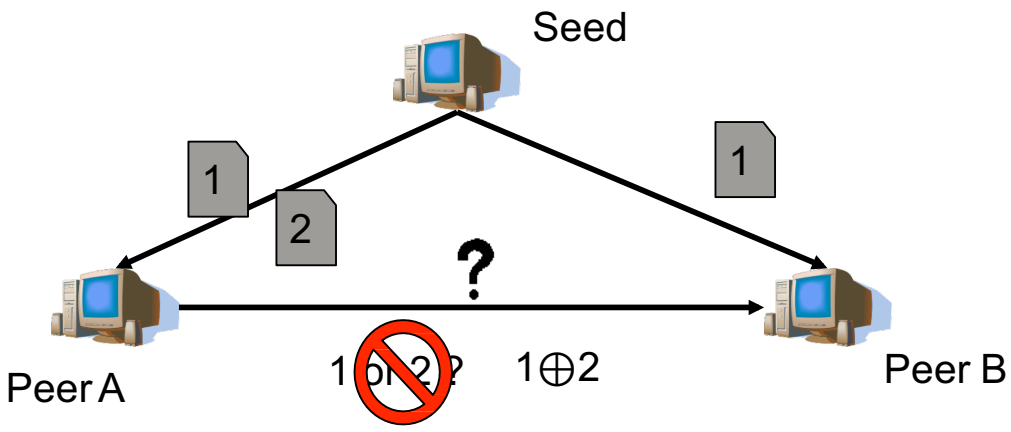
- Original source creates redundant blocks
  - Reed-Solomon, Tornado, Digital Fountain codes, …
- Original file has N blocks, source creates K blocks
- Generally applies:
  - Any N out of N+K are sufficient to reconstruct file
    - Some coding schemes require a bit more than N
- Problem: Redundant data generated by one entity
- Problem: K is fixed
  - Some codes allow for "infinite K"

Network Coding

- How about letting peers do the coding?
- Solution: Network coding
- Idea:
  - Peers create linear combinations of blocks they have
  - Send block with coefficients to others
  - Sufficient number of linearly independent blocks allows for reconstruction of file

Network Coding

- Peer A sends a linear combination of blocks 1 and 2
- Peer B already has block 1, can compute block 2



Network Coding: Math

- File has N blocks
  - For example F = [x1, x2] (two blocks)
- Pick two numbers $a_{i,1}, a_{i,2}$
- Code $E_i (a_{i,1}, a_{i,2}) = a_{i,1} * x_1 + a_{i,2} * x_2$
  - Can create an infinite number of E_i's
- When we have two linearly independent $E_i (a_{i,1}, a_{i,2})$'s, we can re-create original blocks x_1 and x_2
  - Same as solving a set of linear equations
- Note: All math in some finite field, e.g., GF(2^{16})

Practical Considerations

- In principle, information is spread out better
- Need for linear independence
  - May need to download more data than file has
  - No analysis of relevance of this done yet
- Performance has been found to be good
  - Download times shorter and more predictable
- System very robust against departures of seeds
- Does not need altruistic users to stay logged on

Peer-to-Peer Multicast and Streaming

- Streaming content delivery gaining importance
  - Mainly for video-on-demand systems
- Many systems for P2P streaming or multicast

	One recipient	Multiple recipients
Bulk data	Traditional unicast	Multicast
Media data	Unicast streaming	Multicast streaming

Why Multicast?

- Multicast is sending data to many receivers at the same time
- Compare: Broadcast is sending data to everybody
- Two-kinds of multicasting schemes:
- *1-to-m multicast*
  - One sender, many receivers, e.g., video streaming
- *n-to-m multicast*
  - Many senders, many receivers, e.g., video conferencing
- For content distribution 1-to-m multicast is more relevant (and more researched)
- Multicast has two main components:
  - Group management
  - Routing

Multicast Group Management and Routing

- Group is another name for the receivers
- Group management is about how and who manages the membership in the group
  - Centralized, de-centralized, …
- Multicast routing is how to route the packets
- Multicast property: Ideally, each packet will go over any given network link at most once
- Achieved with a multicast tree, rooted at the sender
- Requires support from the routers in the network
- IPv4 has multicast address block (block D, 224.x.x.x)
  - Support from routers not guaranteed
- IPv6 should have native support for multicast

Why Application Layer Multicast (ALM)?

- IP-level multicast is efficient, but requires support from all the routers in the network
- Most routers do not have multicast support turned on
- Result: No Internet-wide multicast possible
  - mbone overlay network offers some support
- Application level multicast performs multicasting on the application level
- Benefit: No support from routers needed
- Disadvantage: Not as efficient as native multicast
- ALM builds an overlay network
- Overlay typically consists of the receivers and senders
  - Some overlays assume fixed nodes in network

ALM Details

- ALM can be classified according to:
  - How they build distribution tree?
    - First mesh, then tree, or directly as spanning tree?
  - How to build overlay?
    - Use a structured network, e.g., DHT as basis?
    - Build directly on top of end-user machines?

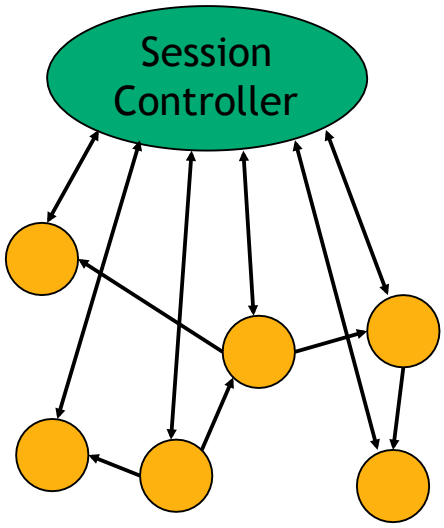
- ALM metrics:
  - Relative Delay Penalty: How much does the one-way delay increase because of overlay (ratio of overlay/direct path)
  - Throughput
  - Stress: How many times a packet traverses a given link
  - Control overhead

ALM Architectures

- Architectures based on unstructured overlays
  - Centralized architecture
  - De-centralized architecture
- Architectures based on structured overlays
  - Flooding-based multicast
  - Tree-based multicast
- Structured overlays are DHTs in this case

Application Level Multicast Infrastructure

- Also known as ALMI
- ALMI is a centralized, unstructured overlay system
- One session controller
- controls the distribution
  - Not part of actual data transfer, just control
- Session controller
- calculates optimal multicast tree



ALMI Details

- When a node wants to join the tree, it contacts the session controller
- Node is assumed to know address of session controller
  - For example, well-known URL, etc.
- Session controller assigns ID to node and gives information about where the node is in the tree
- Joining node contacts its parent in the tree
  - Parent adds new node as child
- When node leaves, it informs the session controller

ALMI Pros and Cons

- Advantages
  - High control over overlay
  - Easy to implement
  - Easy to detect malicious nodes
    - Because of centralized nature

- Disadvantages
  - Poor scalability
  - Session controller is a single point of failure
    - Backup controllers monitor main controller
    - When main controller fails, one of the backups takes over

End System Multicast (ESM)

- Fully distributed ALM scheme based on end nodes
- Group management handled by Narada protocol
- Idea of Narada:
  - Overlay nodes organize themselves in a mesh
  - Source-specific multicast trees are built on top of mesh
- Hence, quality of multicast depends on quality of mesh
- Goal of Narada is to match overlay requirements with underlay quality
  - Meaning: Overlay link should have same properties as the corresponding underlay link between the same nodes
- Limited out-degree of nodes to limit load

Narada Group Management

- Every node participates in group management
- Does not scale well, but Narada's target is small to medium sized multicast networks
- Joining node contacts some existing members
  - Again, addresses assumed to be somehow known
- Every member sends periodic heartbeats messages
- Heartbeats announce e.g., arrival of new node
- Eventually, all nodes know about the new node
- Departures detected by missing heartbeats
- Can also repair network partitions:
  - If heartbeats missing, then send probe to that node
  - If node answers, then reconnect overlay
  - Otherwise assume node is gone

Narada Performance

- Relative Delay Penalty
  - For longer underlay delays, RDP between 2 and 4
  - For short underlay delays, RDP up to 14
  - Explanation: For short underlay links, even a slightly suboptimal overlay will lead to high RDP
- Stress
  - Narada limits maximum stress to 9
  - Sending everything over unicast has maximum stress 128

Flooding-Based Replication

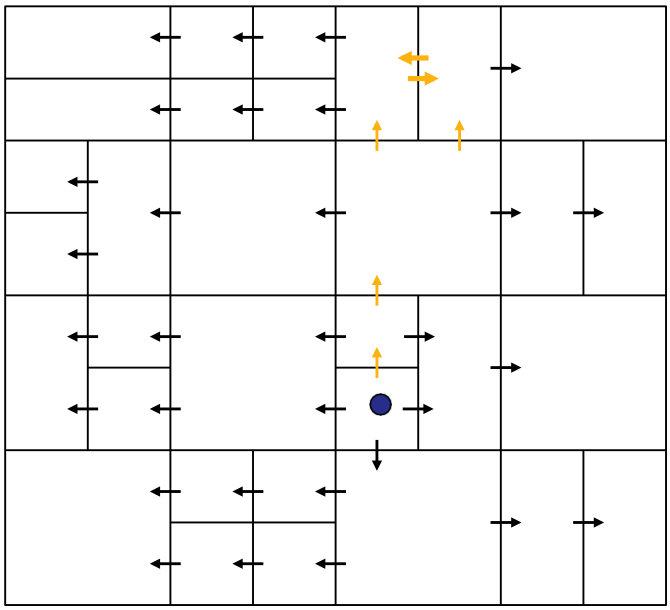
- Use a structured overlay to construct multicast trees
- This case: Use Content Addressable Network, CAN
- Multicast in CAN works as follows
  - 1. Multicast group has a well-known identifier which is mapped to some point in CAN
  - 2. When node wants to join group, it contacts the responsible node for the group ID
  - 3. Responsible node redirects new node to a mini-CAN
    - Mini-CAN has only nodes who are members of multicast group

CAN Multicast Rules

- 1. Source forwards message to all its neighbors
- 2. When node receives message, it forwards it to all its neighbors, except the ones in the direction from which the message came
- 3. If message has traversed more than half of the distance across any dimension, message is not forwarded
- 4. Nodes cache sequence numbers of messages and discard any duplicates they see
  - Rules 1 and 2 ensure messages reach everywhere
  - Rule 3 prevents routing loops
  - Rule 4 is just performance enhancement

CAN Multicast Example

- Orange arrows lead to duplicates
- Note: Not all arrows shown in figure



Tree-Based Example

- Tree-based multicast with structured overlays
- Example: Scribe
  - Based on Pastry DHT
- Idea: Multicast group ID maps to a node in DHT
- That node is root of the multicast tree
  - Also called rendezvous point (RP)
- When a node joins, it sends message to RP
- Message routed in DHT
- Every node on the path checks if it knows about that multicast group
  - If yes, then add new node as child
  - If not, add new node, forward message towards RP

Scribe: Sending Data

- When a node wants to send data to a multicast group, it sends it to the rendezvous point
- Join messages have created the multicast tree rooted at the rendezvous point
- RP forwards the message to its children, who forward it to
- their children, etc.

Comparison of CAN and Scribe

- Relative Delay Penalty
- CAN has higher RDP, typically between 6 and 8
- Scribe has RDP about 2 Stress
- Average stress a bit lower in CAN
- Maximum stress in CAN much lower

- Scribe better for delay-sensitive applications
  - For example, video conference
- CAN better for non delay-critical applications
  - For example, TV broadcast

Conclusions

- BitTorrent is an extremely efficient tool for content distribution
  - Strong incentive system based on game theory
  - Most popular file sharing client since 2001
  - More active users than YouTube and Facebook combined
- However, BitTorrent is a large system with many different mechanisms
  - Ample room to modify the client, alter behavior
  - Cheating can happen, not all strategies are fair