Where we are

Application: (So far: HTTP, Email, DNS)
Today: P2P systems, Overlay Networks

Transport: end-to-end connections, reliability

Network: routing

Link (data-link): framing, error detection

Physical: 1’s and 0’s/bits across a medium (copper, the air, fiber)
Overlay Network (P2P)

- A network made up of “virtual” or logical links
- Virtual links map to one or more physical links
Overlay Network (P2P)

- A network made up of “virtual” or logical links
- Virtual links map to one or more physical links
In our P2P examples with no central server, what’s the best mechanism to find content?

A. Flooding each node and querying
B. Maintaining an entire list at each node
C. Some other system that scales
In our P2P examples with no central server, how would we maintain a mapping of content to nodes?

A. Flooding each node and querying
B. Maintaining an entire list at each node
C. Some other system that scales - a Distributed Hash Table.
Unstructured Overlay Networks

- Overlay links form random graphs
- No defined structure
- Examples: Gnutella: links are peer relationships
Unstructured Overlay Issues

What if the file is rare or far away?

- Search is broken
  - High overhead
  - No guarantee it will work

Redundancy

Traffic Overhead
Structured Overlay Networks (i.e. getting rid of that bit-torrent server…)

• Distribute the tracker information using a Distributed Hash Table (DHT)

• A DHT is a lookup structure
  – Maps keys to an arbitrary value.
  – Works a lot like, well...a hash table.
Recall: Hash Function

- Mapping of any data to a hash value
- if keys are integers, with n nodes in the network
  - \( id = \text{key} \mod n \)
  - E.g., md5sum, sha1, etc.
  - md5: 04c3416cadd85971a129dd1de86cee49

- With a good (cryptographic) hash function:
  - Hash values very likely to be unique
  - Near-impossible to find collisions (hashes spread out)
Distributed Hash Table (DHT)

• DHT: a distributed P2P database
  – Data items stored by a network of peers

• DHT abstraction:
  – Input: key
  – Output: node that stores the content

• Same interface as standard HT: (key, value) pairs
  – get(key) – send key to DHT, get back value
  – put(key, value) – modify stored value at the given key
DHT Goals

• **Scalability**: each node does not keep much state

• **Performance**: small look up latency

• **Load balancing**: no node is overloaded with a large amount of state

• **Dynamic re-configuration**: when nodes join and leave the amount of state moved amongst nodes is minimal

• **Distributed**: no node is more important than others
Distributed Hash Table

- Used in the real world
  - BitTorrent tracker implementation
  - Content distribution networks
  - Many other distributed systems including botnets
DHT: Strawman approach

• Suppose all the keys are integers
• The number of nodes in the network is n
  – id = key % n
DHT: Strawman approach:

- Node 2 dies
- A large number of data items need to be rehashed
  - id = key % n
DHT: Consistent Hashing

• Consistent hashing:
  – hash node -> identifier space
  – hash key -> identifier space

• Node is responsible for a range of keys
  – Multiple key-value pairs assigned to each node

• A key is stored at a node whose identifier is closest to the key in the identifier space

• All DHTs implement consistent hashing

• They differ in the underlying “geometry”
Challenges

• How do we assign (key, value) pairs to nodes?

• How do we find them again quickly?

• What happens if nodes join/leave?
Circular DHT Overlay

• Hash both node ID and key into an m-bit one-dimensional circular identifier space

• Example: 4-bit identifier space [0 – 15]
  – Convert each content key to an integer [0-15] via hash.
  – Convert each node ID to an integer [0 – 15] via hash.
  – The key is stored at its successor: node with next highest integer
Circular DHT Overlay

- Simplest form: each node *only* aware of immediate successor and predecessor.
Circular DHT Overlay

- Simplest form: each node *only* aware of immediate successor and predecessor.
Circular DHT Overlay

- Example: Node 1 wants key “Led Zeppelin IV”
  - Hash the key “Led Zeppelin IV”

Hash both node id and key onto one-dimension circular identifier space
Circular DHT Overlay

- Example: Node 1 wants key “Led Zeppelin IV”
  - Hash the key “Led Zeppelin IV” = 6

Hash both node id and key onto one-dimension circular identifier space

The key is stored at its successor: node with next highest integer
Circular DHT Overlay

- Example: Node 1 wants key “Led Zeppelin IV”
  - Hash the key “Led Zeppelin IV” = 6
Circular DHT Overlay

- Example: Node 1 wants key “Led Zeppelin IV”
  - Hash the key “Led Zeppelin IV” = 6
Circular DHT Overlay

• Example: Node 1 wants key “Led Zeppelin IV”
  – Hash the key “Led Zeppelin IV” = 6
Circular DHT Overlay

• Example: Node 1 wants key “Led Zeppelin IV”
  – Hash the key “Led Zeppelin IV” = 6

If anybody has it, it’s my successor.
Example: Node 1 wants key “Led Zeppelin IV”
   – Hash the key “Led Zeppelin IV” = 6
Circular DHT Overlay

- Example: Node 1 wants key “Led Zeppelin IV”
  - Hash the key “Led Zeppelin IV” = 6
Peer/Node churn

Handling node churn:
- nodes may come and go (churn)
- each node knows address of two of its successors
- each node periodically pings its two successors to check aliveness
- if immediate successor leaves, choose next successor as new immediate successor
Example: node 5 abruptly leaves

- Node 4 detects peer 5 departure;
- makes 8 its immediate successor;
- asks 8 who its immediate successor is;
- makes 8’s immediate successor its second successor.
Tapestry/Pastry

- Node IDs are numbers in a ring
  - 128-bit circular ID space
- Node IDs chosen at random
- Messages for key \( X \) is routed to live node with longest prefix match to \( X \)
  - Incremental prefix routing
  - \( 1110: \)
    - 1XXX \( \rightarrow \) 11XX \( \rightarrow \) 111X \( \rightarrow \) 1110

Slide 35
Summary of DHT Overlays

- A namespace
  - For most, this is a linear range from 0 to $2^{160}$
- A mapping from key to node
  - Chord: keys between node X and its predecessor belong to X
  - Tapestry/Pastry: keys belong to node w/ closest identifier
  - Dynamo: Amazon’s Highly Available Key-value Store
High-Performance Content Distribution

• Problem:
  You have a service that supplies lots of data. You want good performance for all users!

(often “lots of data” means media files)
What is a Content Distribution Network?

An overlay network, that is geo-distributed and stores cached content “close” to users.

At least 70% of the world’s bits are delivered by a CDN!
Content distribution networks (CDNs)

- CDN: stores copies of content (e.g. MADMEN) at CDN nodes
Examples of CDNs

• Akamai
  – 147K+ servers, 1200+ networks, 650+ cities, 92 countries
• Limeligh
  – Well provisioned delivery centers, interconnected via a private fiber-optic connected to 700+ access networks
• Edgecast
  – 30+ PoPs, 5 continents, 2000+ direct connections
• Others
  – Google, Facebook, AWS, AT&T, Level3
CDN caching

• Locality of reference:
  – Users tend to request the same object in succession
  – Some objects are popular: requested by many users
Key Components of a CDN

• Distributed servers
  – Usually located inside of other ISPs
  – Often located in IXPs
• High-speed network connecting them
• Clients (eyeballs)
  – Can be located anywhere in the world
  – They want fast web performance
• Glue
  – Something that binds clients to “nearby” replica servers
High-Performance Content Distribution

• Major challenges:
  – How do we direct the user to a nearby replica instead of the centralized source?
  – How do we determine which replica is the best to send them to?
  – Ensure that replicas are always available?
Challenge 1: Finding the CDN

- Three main options:
  - Application redirect (e.g., HTTP)
  - “Anycast” routing
  - DNS resolution (most popular in practice)

- Example: CNN + Akamai
CNN + Akamai

Request: cnn.com/article
Response: HTML with link to cache.cnn.com media

Content servers: serve media.
CNN + Akamai

Request: cnn.com/article
Response: HTML with link to cache.cnn.com media

Content servers: serve media.
CNN + Akamai

Request: cnn.com/article
Response: HTML with link to cache.cnn.com media

Content servers: serve media.

Akamai’s DNS response directs user to selected server.

Retrieved media file.
CNN + Akamai

Request: cnn.com/article
Response: HTML with link to cache.cnn.com media

Akamai’s DNS response directs user to selected server.

Content servers: serve media.
Content in today’s Internet

• Most flows are HTTP
  – Web is at least 52% of traffic
  – Median object size is 2.7K, average is 85K (as of 2007)

• Is the Internet designed for this common case?
  – Why?
Why speed matters

• Impact on user experience
  – Users navigating away from pages
  – Video startup delay

• 4x increase in abandonment with 10s increase in delay
Streaming Media

- Straightforward approach: simple GET

- Challenges:
  - Dynamic network characteristics
  - Varying user device capabilities
  - User mobility
Dynamic Adaptive Streaming over HTTP (DASH)

• Encode several versions of the same media file
  – low / medium / high / ultra quality

• Break each file into chunks

• Create a “manifest” to map file versions to chunks / video time offset
Dynamic Adaptive Streaming over HTTP (DASH)

- Client requests manifest file, chooses version
- Requests new chunks as it plays existing ones
- Can switch between versions at any time!
Summary

• Peer-to-peer architectures for:
  – High performance: BitTorrent
  – Decentralized lookup: DHTs

• CDNs: locating “good” replica for media server

• DASH: streaming despite dynamic conditions