15-441/641: Content Delivery and Peer-to-Peer

15-441 Fall 2019

Profs **Peter Steenkiste** & Justine Sherry

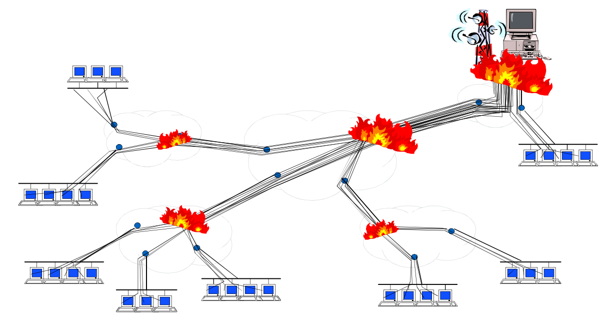
Fall 2019

https://computer-networks.github.io/fa19/

Carnegie Mellon University

Problem: Scaling Content Delivery

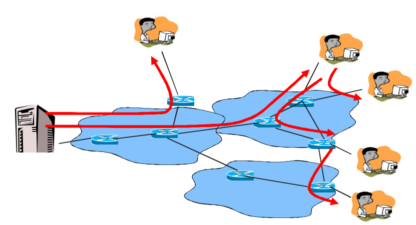
- Millions of clients ⇒ server and network meltdown



Outline

- Peer-to-peer
  - Overlays: naming, addressing, and routing
- CDNs
- Load balancing – consistent hashin

P2P System



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

P2P Definition

Distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority.

– A Survey of Peer-To-Peer Content Distribution Technologies, Androutsellis-Theotokis and Spinellis

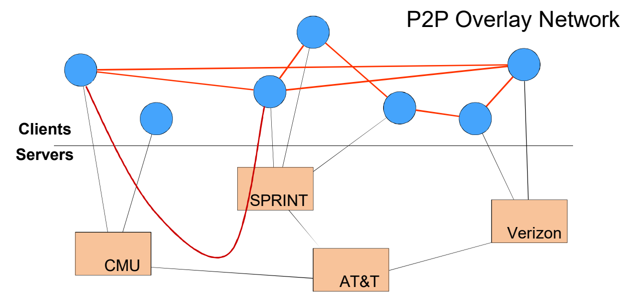
Why p2p?

- Harness lots of spare capacity
  - 1 Big Fast Server: $10k/month++ versus 1000s .. 1000000s clients: $??
  - Capacity grows with the number of users!
- Build very large-scale, self-managing systems
  - Same techniques useful for companies,
    - E.g. Akamai's 14,000+ nodes, Google's 100,000+ nodes
    - But: servers vs. arbitrary nodes, hard vs. soft state (backups vs caches), ….
    - Also: security, fairness, freeloading, ..
- No single point of failure
  - Some nodes go down – others take over
  - … government shuts down nodes – peers in other countries are available

Key Idea: Network Overlay

- A network overlay is a network that is layered on top of the Internet
  - Simplified picture: overlays use IP as their datalink layer
- Overlays need the equivalent of all the functions IP networks need:
  - Naming and addressing
  - Routing
  - Bootstrapping
  - Security, error recovery, etc.

P2P Construction



Names, addresses, and routing

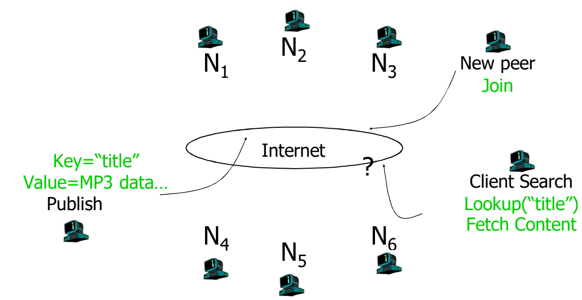
The Internet

- Endpoint: host
- Name: hierarchical domain name
- Address: IP address of node that has the content, plus content name
- Routing: how to reach host, e.g., BGP, …

Content retrieval:

- End-point: content
- Name: identifies content you are looking for
  - E.g., hash of file, key words
- Address: the IP address of node that has the content, plus content name
- Routing: how to find the data

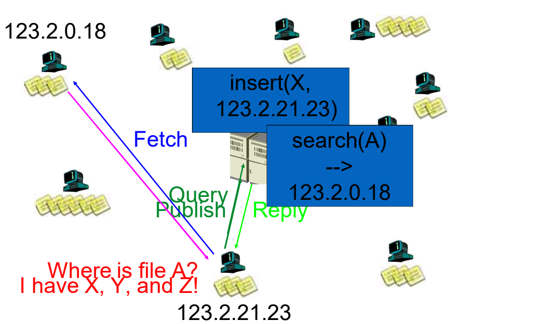
Common P2P Framework



What is (was) out there?

	Central	Flood	Super-node flood	Route
Whole File	Napster	Gnutella		Freenet
Chunk Based	BitTorrent		KaZaA (bytes, not chunks)	DHTs eDonkey 2000

Napster: Central Database

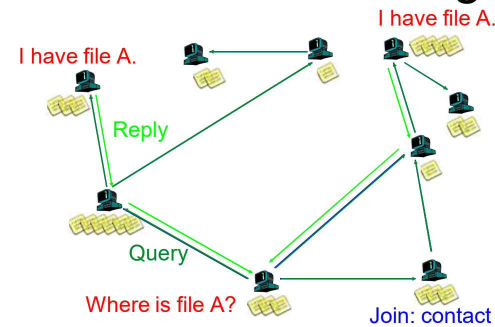


Join: contact server

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

Gnutella: Flooding



Join: contact peers
Publish: noop
Fetch: direct p2p

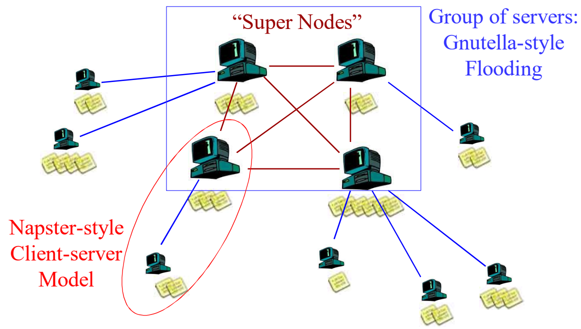
Gnutella: Discussion

- Pros:
  - Fully de-centralized
  - 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
- TTL-limited search works well for haystacks.
  - For scalability, does NOT search every node.
  - May have to re-issue query later

KaZaA: Query Flooding

- First released in 2001 and also very popular
- **Join**: on startup, client contacts a “supernode” ... may at some point become one itself
- **Publish**: send list of files to supernode
- **Search**: send query to supernode, supernodes flood query amongst themselves.
- **Fetch**: get the file directly from peer(s); can fetch simultaneously from multiple peers

KaZaA: Intelligent Query Flooding



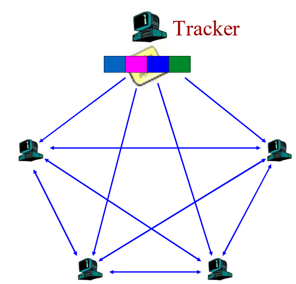
KaZaA: Discussion

- Works better than Gnutella because of query consolidation
- Several nodes may have requested file... How to tell?
  - Must be able to distinguish identical files
  - Same filename not necessarily same file...
- Use Hash of file
  - Can fetch bytes [0..1000] from A, [1001...2000] from B
- Pros: Tries to take into account node heterogeneity:
  - Bandwidth, computational resources, …
- Cons: Still no guarantees on search scope or time
- Challenge: want stable superpeers – good prediction
  - Must also be capable platforms

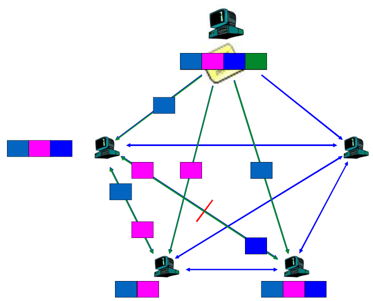
BitTorrent: Swarming

- Started in 2001 to efficiently support flash crowds
  - Focus is on fetching, not searching
- **Publish**: Run a tracker server.
- **Search**: Find a tracker out-of-band for a file, e.g., Google
- **Join**: contact central "tracker" server for list of peers.
- **Fetch**: Download chunks of the file from your peers. Upload chunks you have to them.
- Comparison with earlier architectures:
  - Focus on fetching of "few large files"
  - Chunk based downloading
  - Anti-freeloading mechanisms

BitTorrent: Publish/Join



BitTorrent: Fetch



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
  - (Tracker is a design choice, not a requirement, as you know from your projects. Could easily combine with other approaches.)

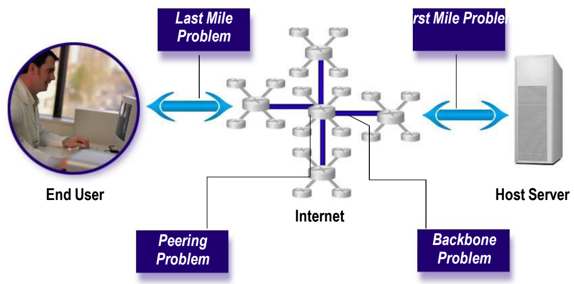
When are p2p Useful?

- Works well for caching and “soft-state”, read-only data
  - Works well! BitTorrent, KaZaA, etc., all use peers as caches for hot data
- Difficult to extend to persistent data
  - Nodes come and go: need to create multiple copies for availability and replicate more as nodes leave
- Not appropriate for search engine styles searches
  - Complex intersection queries (“the” + “who”): billions of hits for each term alone
  - Sophisticated ranking: Must compare many results before returning a subset to user
  - Need massive compute power

Outline

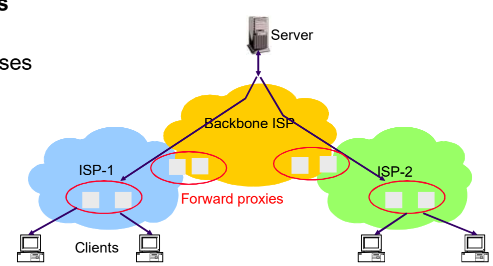
- Peer-to-peer
  - Overlays: naming, addressing, and routing
- CDNs
- Load balancing – consistent hashin

Content Delivery: Possible Bottlenecks



Reminder: Caching with Forward Proxies

- Cache documents close to **clients** →decrease latency
- Typically done by ISPs or enterprises →reduce provider traffic load
- CDNs proactively cache for the content providers (their clients)
- Typically cache at different levels in the Internet hierarchy:
  - Last mile ISPs for low latency
  - Closer to core for broader coverage

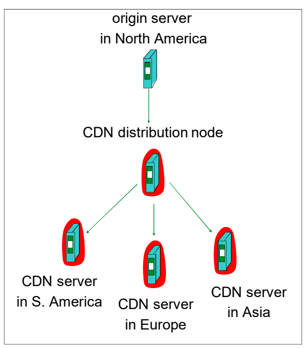


Content Distribution Networks (CDNs)

- The content providers are the CDN customers.

Content replication

- CDN company installs hundreds of CDN servers throughout Internet
  - Close to users
- CDN replicates its customers' content in CDN servers. When provider updates content, CDN updates servers



31

What is the CDN?

- Edge Caches: work with ISP and networks everywhere to install edge caches
  - Edge = close to customers
- Content delivery: getting content to the edge caches
  - Content can be objects, video, or entire web sites
- Mapping: find the "closest" edge server for each user and deliver content from that server
  - Network proximity not the same as geographic proximity
  - Focus is on performance as observed by user (quality)

15-441 S'10

32

Potential Benefits

- Very good scalability
  - Near infinite if deployed properly
- Good economies at large scales
  - Infrastructure is shared efficiently by customers
  - Statistical multiplexing: hot sites use more resources
- Can reduce latency – more predictable performance
  - Through mapping to closest server
  - Avoids congestion and long latencies
- Can be extremely reliable
  - Very high degree of redundancy
  - Can mitigate some DoS attacks

Server Selection

- Which server?
  - Lowest load: to balance load on servers
  - Best performance: to improve client performance
    - Based on Geography? RTT? Throughput? Load?
  - Any alive node: to provide fault tolerance
- How to direct clients to a particular server?
  - As part of naming: DNS redirect
  - As part of application: HTTP redirect
  - As part of routing: anycast, cluster load balancing

Finding the "Closest Edge Cache – Example: Akamai DNS Redirect

- Akamai creates new domain names for each client
  - e.g., *a128.g.akamai.net* for *cnn.com*
- The CDN's DNS servers are authoritative for the new domains
- The client content provider modifies its embedded URLs (= <u>names</u>) to reference the new domains – "Akamaize" content
  - e.g.: *http://www.cnn.com/image-of-the-day.gif* becomes
  - *http:// a128.g.akamai.net/image-of-the-day.gif* – *<u>name</u> in the overlay*
- Requests now sent to CDN's infrastructure…
- Generates and <u>address</u>: IP address of server + URI (tuple)
- <u>Routing</u> inside Akamai system identifies right replica to route to
  - IP takes care of rest once a replica has been selected (overlay!)

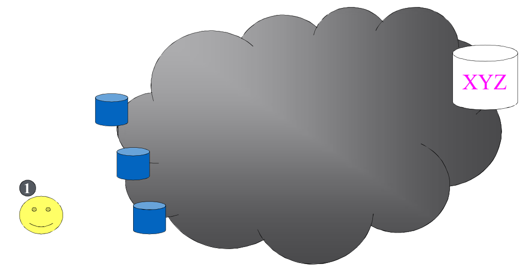
Effectively another layer of routing: the path your connection takes is redirected using DNS.

Alternative Approaches

- Routing based (IP anycast)
  - Multiple CDN instances advertise the same IP address block
  - BGP will route packets to the closest one (fewest AS hops)
  - Pros: Transparent to clients, works when browsers cache failed addresses, circumvents many routing issues
  - Cons: Little control, complex, scalability, TCP can't recover
- Application based (HTTP redirects)
  - Send request to origin HTTP server which redirects the HTTP request to a CDN instance closer to the client
  - Pros: Application-level, fine-grained control
  - Cons: Additional load and RTTs, hard to cache, availability concerns

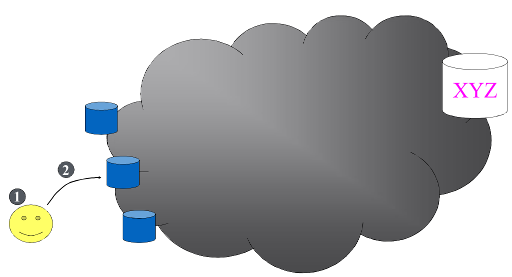
17

Process Flow



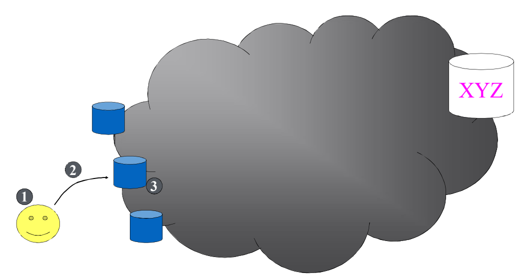
1. User wants to download distributed web content

Process Flow



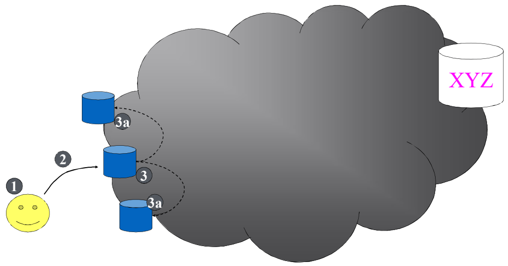
2. User is directed through Akamai's dynamic mapping to the "closest" edge cache

Process Flow



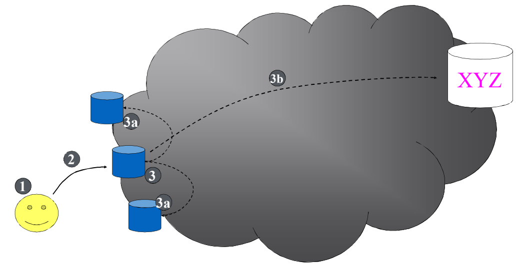
3. Edge cache searches local hard drive for content

Process Flow



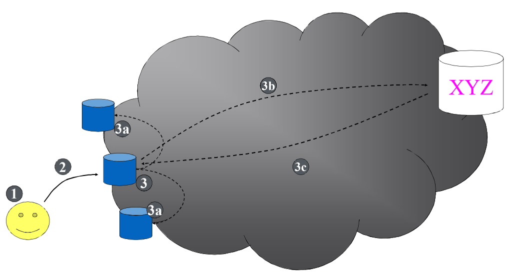
3b. If requested object is not on local hard drive, edge cache checks other edge caches in same region for object

Process Flow



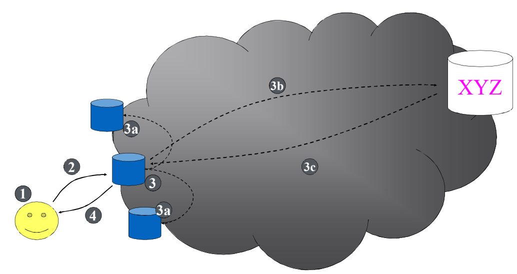
3b. If requested object is not cached or not fresh, edge cache sends an HTTP GET the origin server

Process Flow



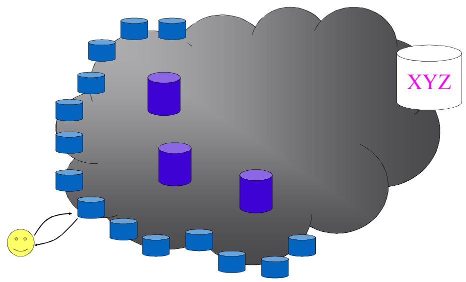
3c. Origin server delivers object to edge cache over optimized connection

Process Flow



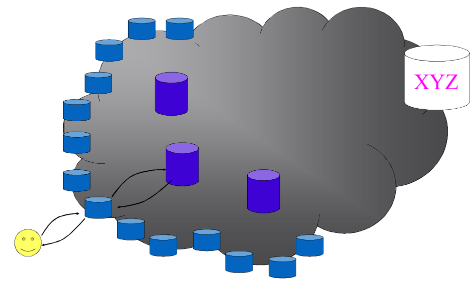
4. Edge server delivers content to end user

Core Hierarchy Regions



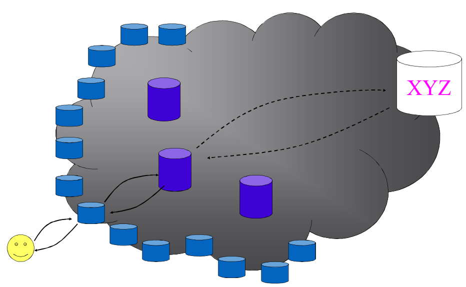
1. User requests content and is mapped to optimal edge Akamai server

Core Hierarchy Regions



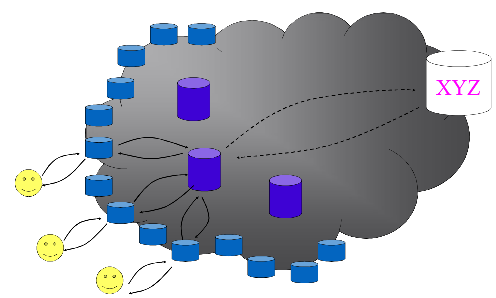
2. If content is not present in the region, it is requested from most optimal core region

Core Hierarchy Regions



3. Core region makes one request back to origin server

Core Hierarchy Regions



4. Core region can serve many edge regions with one request to origin server

Core CDN Features

Reduces traffic back to origin server

- Reduces infrastructure needs of customer
- Provides best protection against flash crowds
  - Especially important for large files (*e.g.* Operating System updates or video files)

Improved end-user response time

- Core regions are well connected
- Optimized connection speeds object delivery

Outline

- Peer-to-peer
  - Overlays: naming, addressing, and routing
- CDNs
- Load balancing – consistent hashin

Distributing Load across Servers

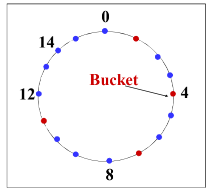
- Given document XYZ, we need to choose a server to use
  - E.g., in a data center
- Suppose we use simple hashing: modulo n of a hash of the name of the document
- Number servers from 1…n
  - Place document XYZ on server (XYZ mod n)
  - What happens when a servers fails? n → n-1
    - Same if different people have different measures of n
  - Why might this be bad?

Consistent Hash: Goals

- “view” = subset of all hash buckets that are candidate locations
  - Correspond to a real server
- Desired features
  - Load – all hash buckets have a similar number of objects assigned to them
  - Smoothness – little impact on hash bucket contents when buckets are added/removed
  - Spread – small set of hash buckets that may hold an object regardless of views

Consistent Hash – Example

- Construction
  - Assign each of C hash buckets to random points on mod 2^n circle, where, hash key size = n.
  - Map object to random position on unit interval
  - Hash of object = closest bucket
- Monotone → addition of bucket does not cause movement between existing buckets
- Spread & Load → small set of buckets that lie near object
- Balance → no bucket is responsible for large number of objects

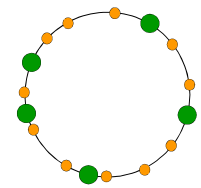


Consistent Hashing: Ring

- Use consistent has to map both keys and nodes to an m-bit identifier in the same (metric) identifier space
  - For example, use SHA-1 hashes
  - **Node identifier:** SHA-1 hash of IP address

    IP="198.10.10.1" $\xrightarrow{\text{SHA-1}}$ ID=123
  - **Key identifier:** SHA-1 hash of key

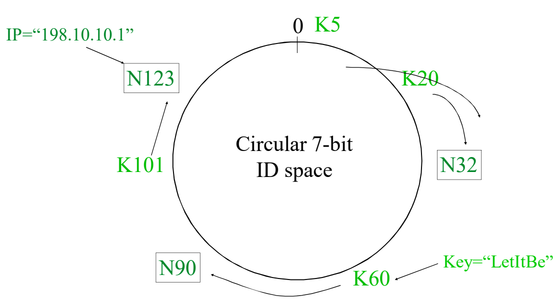
    Key="LetItBe" $\xrightarrow{\text{SHA-1}}$ ID=60



- Also need "rule" for assigning keys to nodes
  - For example: "closest", higher, lower, ..

Consistent Hashing Example

Rule: A key is stored at its successor: node with next higher or equal ID



Consistent Hashing Properties

- **Load balance:** all nodes receive roughly the same number of keys
  - For N nodes and K keys, with high probability
    - Each node holds at most $(1+\varepsilon)K/N$ keys
    - Provided that K is large compared to N
- When server is added, it receives its initial work load from "neighbors" on the ring
  - "Local" operation: no other servers are affected
  - Similar property when a server is removed

Finer Grain Load Balancing

- Redirector knows all server IDs s_i
- It can also track approximate "load" for more precise load balancing
  - Need to define load and be able to track it
- To balance load:
  - W_i = Hash(URL, ip of s_i) for all i
  - Sort W_i from high to low
  - Find first server with low enough load
- Benefits and drawbacks?

Consistent Hashing Used in Many Contexts

- Distribute load across servers in a data center
  - The redirector sits in data center
- Finding storage cluster for an object in a CDN uses centralized knowledge
  - Why?
  - Can use consistent hashing in the cluster
- Consistent hashing can also be used in a distributed setting
  - P2P systems can use it find files (DHTs)