

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

15-441 Spring 2019
 Profs **Peter Steenkiste** & Justine Sherry

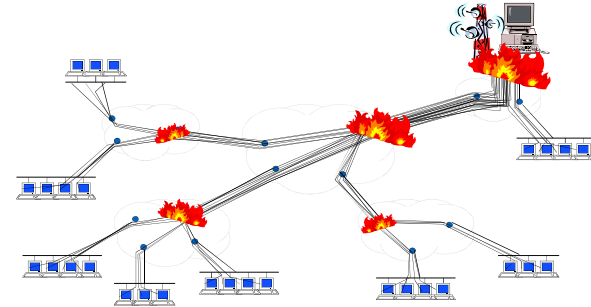
Fall 2019
<https://computer-networks.github.io/sp19/>



**Carnegie
Mellon
University**

Problem: Scaling Content Delivery

- Millions of clients \Rightarrow server and network meltdown



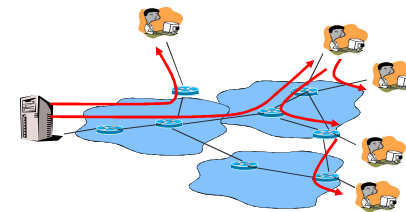
Outline

- Peer-to-peer
 - Overlays: naming, addressing, and routing
- CDNs



3

P2P System



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



4

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

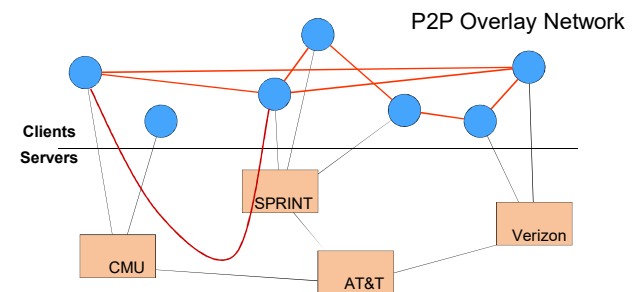


Key Idea: Network Overlay

- 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

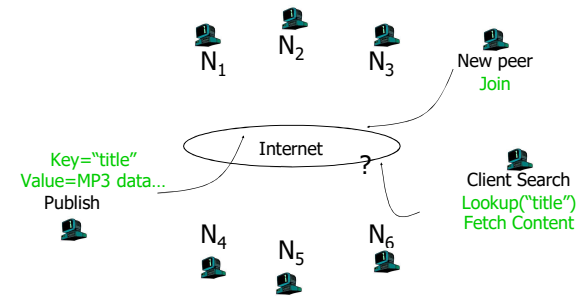
- Endpoint: host
- Name: hierarchical domain name
- Address: IP address of node that has the content, plus content name
- Routing: how to reach host, e.g., BGP, ...

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

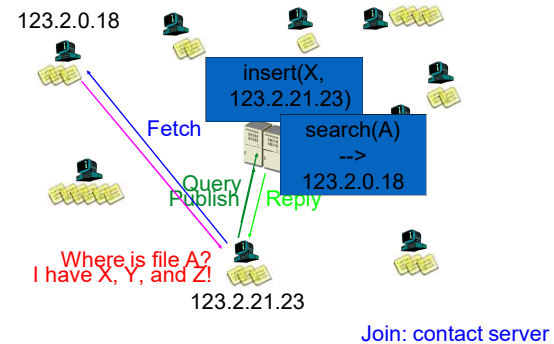


What is (was) out there?

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



Napster: Central Database



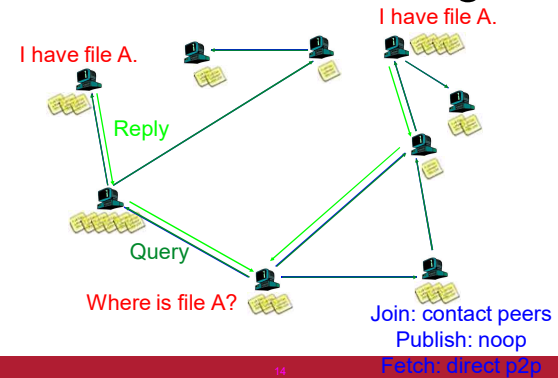
Napster: Discussion

- 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



13

Gnutella: Flooding



14

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



15

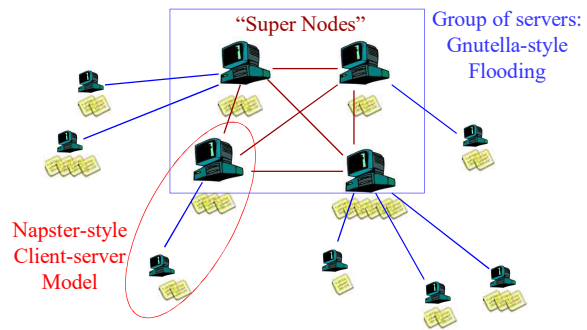
KaZaA: Query Flooding

- 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



16

KaZaA: Intelligent Query Flooding



17

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



20

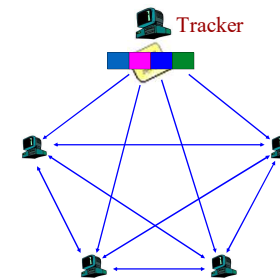
BitTorrent: Swarming

- 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



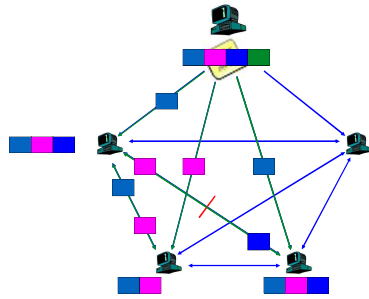
22

BitTorrent: Publish/Join



23

BitTorrent: Fetch



24

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.)



26

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



27

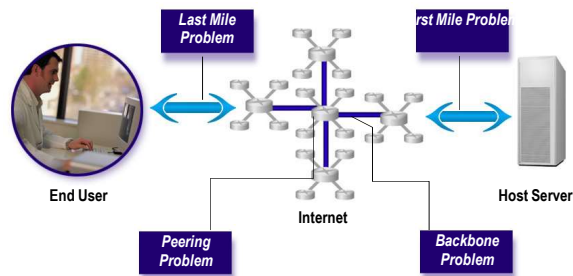
Outline

- Peer-to-peer
- Overlays, naming, ..
- CDNs



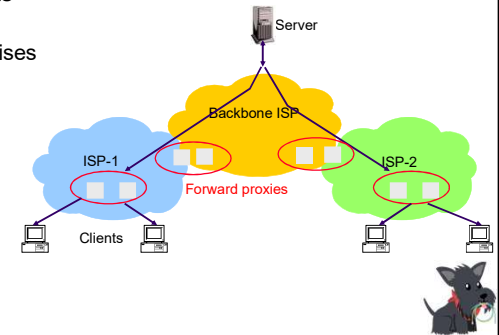
28

Content Delivery: Possible Bottlenecks



Improving HTTP Performance: 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



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)



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



Revisiting CDN 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 (= names) 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](http://a128.g.akamai.net/image-of-the-day.gif) – name in the overlay
- **Requests now sent to CDN's infrastructure...**
- Generates and address: IP address of server + URI (tuple)
- Routing 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 the DNS.



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 described earlier
 - As part of application: HTTP redirect
 - As part of routing: anycast, cluster load balancing



16

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

The diagram shows a grey cloud labeled 'XYZ' with three blue cylinders representing servers. A yellow smiley face with a '1' above it is on the left, with an arrow pointing towards the cloud.

Process Flow

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

The diagram is similar to the first slide, but the smiley face now has a '2' above it and an arrow pointing to the middle server.

Process Flow

3. Edge cache searches local hard drive for content

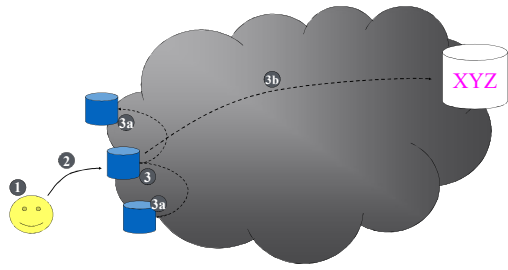
The diagram shows the smiley face with a '3' above it and an arrow pointing to the middle server.

Process Flow

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

The diagram shows the smiley face with a '3' above it and an arrow pointing to the middle server. Two additional servers are shown with dashed circles around them, labeled '3a', representing other edge caches in the region.

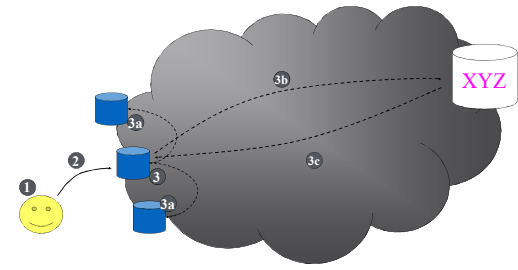
Process Flow



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



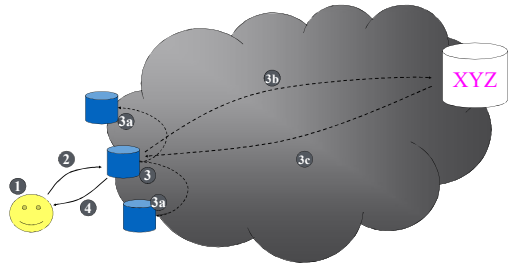
Process Flow



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



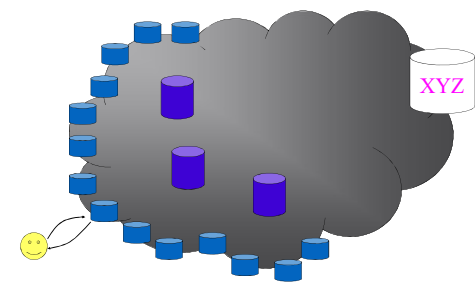
Process Flow



4. Edge server delivers content to end user



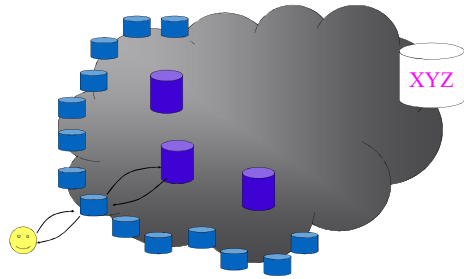
Core Hierarchy Regions



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



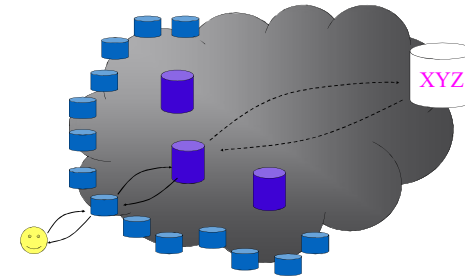
Core Hierarchy Regions



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



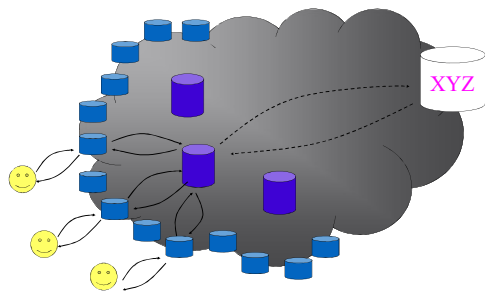
Core Hierarchy Regions



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

