Overlay and P2P Networks
Applications

Prof. Sasu Tarkoma
10.2.2015 and 12.2.2015
Contents

• Monday 9.2. Applications I
  • BitTorrent Mainline DHT
  • Scribe and PAST
• Thursday 12.2. Applications II
  • P2PSIP
  • Internet Indirection Architecture
  • Content Delivery Networks (CDN)
  • Coral CDN
  • CDN Federation
  • P2P CDN
• Monday 16.2.
  • Dynamo and NoSQL
  • SDN and Clouds
• Thursday 19.2.
  • Advanced topics and summary
Consistent hashing alleviates network problems and eventual consistency can be achieved through replication and synchronization.

Example: Dynamo

Replication, Gossip, etc.

Selective flooding

Consistent hash (O(1) DHT)

Search | Storage | Rendezvous

Good for arbitrary data and search functions, can aggregate routing info, structure improves scalability.

Examples: Gnutella and Freenet

Example: BitTorrent

Limited flooding / depth first / Bloom filters

Tracker

DHT

Search | Storage | Rendezvous

Good for name/value data, note flat address space, one node is responsible, churn is a concern.

Examples: Lookup: Chord, CAN, Kademlia

Storage: PAST

Rendezvous: Scribe (for multicast), i3

Cluster

Wide-area (unstructured)

Wide-area (structured)

Internet (TCP/IP)

Consistent hashing alleviates network problems and eventual consistency can be achieved through replication and synchronization.
BitTorrent Mainline DHT

Decentralized tracker (trackerless torrent)

Based on Kademlia

Uses a custom RPC based on UDP

The **key** is the **info-hash**, the hash of the metadata. It uniquely identifies a torrent.

The **data** is a peer list of the peers in the swarm

Torrents have bootstrap nodes in the overlay
BitTorrent Mainline DHT

Each peer announces itself with the distributed tracker.
Looking up the 8 nodes closest to the info-hash of the torrent and sending an announce message to them.

Those 8 nodes will then add the announcing peer to the peer list stored at that info-hash.

A peer joins a torrent by looking up the peer list at a specific info-hash.

Nodes return the peer list if they have it.
Kademlia in Bittorrent Mainline DHT

The implementation extends the single bit model discussed before.

The single bit model can be seen to have a prefix first $n-1$ bits need to match for the $n$th list.

The extension introduces prefix (group of bits)-based operation with width $w$ for digits, giving $2^w - 1$ k-buckets with the missing one containing the node ID.

An $m$-bit prefix reduces the maximum number of lookups from $\log_2 n$ to $\log_2^w n$.

This results in a prefix-based routing table!
Each node knows more about close nodes than distant nodes.

Key space of each bucket grows with the power of 2 with the distance.

Querying for an ID will on average halve the distance to the target in each step.
Comparisons

Kademlia and Chord
   Chord has only one direction on the ring
   Incoming traffic cannot be used to improve routing table
   But Chord has pred/succ (sequential neighbours)

Kademlia and Pastry
   Pastry has more complex table
   Pastry has sequential neighbours

What about Mainline DHT in practice?
Implementation Details

Mainline DHT implements Kademlia with a width of 2, and $k = 8$ nodes in each bucket.

Keys are replicated on the three nodes with nodeID nearest the key with a 30-minute timeout.

If a node fails, the keys will be lost.

Nodes learn implicitly
- Iterative queries, incoming messages
- Lazy removal
- Ping LRU node when bucket full
Reported Problems with Mainline DHT

An Analysis of BitTorrent’s Two Kademlia-Based DHTs
Scott A. Crosby and Dan S. Wallach, 2007

Do the DHTs work correctly? No. Mainline BitTorrent dead-ends its lookups 20% of the time and Azureus nodes reject half of the key store attempts.

What is the DHT lookup performance? Both implementations are extremely slow, with median lookup times around a minute.

Why do lookups take over a minute? Lookups are slow because the client must wait for RPCs to timeout while contacting dead nodes. Dead nodes are commonly encountered in the area closest to the destination key.

Why are the routing tables full of dead nodes? Kademlia’s use of iterative routing limits the ability for a node to opportunistically discover dead nodes in its routing table (refresh, explicit ping)
Design Problems

Iterative search can return dead nodes (no checking)
Recursive routing would implicitly define liveness

Dead nodes are pruned only with refresh or explicit ping

XOR metric
cannot enumerate nodes (as in Pastry or Chord)

Nodes can be ordered based on distance to given key
PAST

PAST: Cooperative, Archival File Storage and Distribution

Runs on top of Pastry, pastry routes to closest live nodeId

Strong persistence, high availability, scalability

API:
  Insert: store replica of a file at k diverse storage nodes
  Lookup: retrieve file from a nearby live storage node
  Reclaim: free storage associated with a file

Files are immutable!
Storage Invariant:
File “replicas” are stored on k nodes with nodeIds closest to fileId

(k is bounded by the leaf set size)
PAST File Retrieval

- Lookup
- fileId
- k replicas

file located in log_{16} N steps (expected)
usually locates replica nearest client C
PAST Features

Caching
On nodes along the route of lookup and insert messages (as in Freenet)
Aim to balance load

Security
No read access control, encryption can be used
File authenticity with certificates
System integrity: ids non-forgable, sign sensitive messages
Randomized routing
SCRIBE: Large-scale, decentralized multicast

Infrastructure to support topic-based publish/subscribe applications

Reasonable performance compared to IP multicast

Publish `topicId`

Subscribe `topicId`
Session Initiation Protocol (SIP)

An Application-layer control (signaling) protocol for creating, modifying and terminating sessions with one or more participants.

Sessions include Internet multimedia conferences, Internet telephone calls and multimedia distribution.

Members in a session can communicate via multicast or via a mesh of unicast relations, or a combination of these.

Text based, model similar to HTTP.
P2P SIP

SIP is already ready for P2P Active standardization in IETF

Uses symmetric, direct client-to-client communication

Intelligence resides mostly on the network border in the user agents
The proxies and the registrar only perform lookup and routing

The lookup/routing functions of the proxies/registrar can be replaced by a DHT overlay built in the user agents.

By adding join, leave and lookup capabilities, a SIP user agent can be transformed into a peer capable of operating in a P2P network
Internet Indirection Infrastructure (i3)

- A DHT-based overlay network
  - Based on Chord
- Aims to provide more flexible communication model than current IP addressing
- Also a forwarding infrastructure
  - i3 packets are sent to identifiers
  - each identifier is routed to the i3 node responsible for that identifier
  - the node maintains triggers that are installed by receivers
  - when a matching trigger is found the packet is forwarded to the receiver
i3 II

• An i3 identifier may be bound to a host, object, or a session
• i3 has been extended with ROAM
  – Robust Overlay Architecture for Mobility
  – Allows end hosts to control the placement of rendezvous-points (indirection points) for efficient routing and handovers
  – Legacy application support
    • user level proxy for encapsulating IP packets to i3 packets
R inserts a trigger (id, R) and receives all packets with identifier id.

Mobility is transparent for the sender

the host changes its address from R1 to R2, it updates its trigger from (id, R1) to (id, R2).
A multicast tree using a hierarchy of triggers

Source: http://i3.cs.berkeley.edu/
Anycast using the longest matching prefix rule.
Sender-driven service composition using a stack of identifiers

(a) Sender-driven service composition

Receiver-driven service composition using a stack of identifiers

(b) Receiver-driven service composition

Source: http://i3.cs.berkeley.edu/
PlanetLab (www.planet-lab.org)

Global research network that support the development of network services
As of Feb 2014, PlanetLab has 1181 nodes at 567 sites
Content Delivery Networks (CDN)

Geographically distributed network of Web servers around the globe (by an individual provider, E.g. Akamai).

Improve the performance and scalability of content retrieval.

Allow several content providers to replicate their content in a network of servers.
Motivation

Network cost
Huge cost involved in setting up clusters of servers around the globe and corresponding increase in network traffic

Economic cost
Higher cost per service rate making them inaccessible to lower and medium level customers
Use cases

Static or slow changing content

Popular content shared by many users

Geographically distributed users

Ad-hoc or irregular usage that does not benefit from local caching

Expensive or saturated bandwidth connections

Flashcrowds
CDN Technology

Intelligent wide area traffic management
Direct clients’ requests to optimal site based on topological proximity

Two types of redirection: **DNS redirection** or **URL rewriting**

Cache
Saves useful contents in cache nodes.

Two cache policies: **least frequently used** standard and **least recently used** standard.
CDN Types

CDNs

Hosting CDN

Relaying CDN

Partial Site Content Delivery

Full Site Content Delivery

Request Routing Techniques

DNS based

URL Rewriting
CDN

Replicate content on many servers

Challenges

How to replicate content
Where to replicate content
How to find replicated content
How to choose among known replicas

How to direct clients towards replica
  DNS, HTTP redirect, anycast, etc.

Akamai
Server Selection

Service and content is replicated in many places in network

How to direct clients to a particular server?
- As part of routing → anycast, cluster load balancing
- As part of application → HTTP redirect
- As part of naming → DNS

Which server to use?
- Best performance → to improve client performance
  - Based on Geography? RTT? Throughput? Load?
- Lowest load → to balance load on servers
- Any active node → to provide availability
CDN Architecture

- Origin Server
- CDN
- Request Routing Infrastructure
- Distribution and Accounting Infrastructure
- Surrogate
- Client
Client ISP

Client DNS
(Local DNS server for client)

CDN DNS

Content Provider

DNS

1

6

1

6

3

redirection

2

4

5
Surrogate Server placement problem

Given N possible locations at edge of the Internet, we are able to place K (K<N) surrogate servers, how to place them to minimize the total cost?

This is the: Minimum K-Median Problem

Given N points, we must select K (centers), and then assign each input point j to the selected center that is closest to it. The goal is to select K centers so as to minimize the sum of the assignment costs.

This is NP-Hard

Note: many cost functions can be used, can take into account clients that generate greatest load
Many heuristics algorithms to solve this (tree, greedy, hot spot...)
<table>
<thead>
<tr>
<th>CDN</th>
<th>Type</th>
<th>Coverage</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Akamai</td>
<td>Commercial</td>
<td>Market leader</td>
<td>Edge platform for handling static and dynamic content, DNS-based request-routing</td>
</tr>
<tr>
<td></td>
<td>CDN service including streaming data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Limelight Networks</td>
<td>Commercial</td>
<td>Surrogate servers in over 70 locations in the world</td>
<td>Edge-based solutions for content delivery, streaming support, custom CDN for custom delivery solutions, DNS-based request-routing</td>
</tr>
<tr>
<td></td>
<td>On-demand distribution, live video, music, games, …</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coral</td>
<td>Academic</td>
<td>Experimental, hosted on PlanetLab</td>
<td>Uses a DHT algorithm (Kademlia), support for static content, DNS-based request-routing</td>
</tr>
<tr>
<td></td>
<td>Content replication based on popularity (on demand), addresses flash crowds</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoDeeN</td>
<td>Academic testbed</td>
<td>Experimental, hosted on PlanetLab, collaborative CDN</td>
<td>Support for static content, HTTP direction</td>
</tr>
<tr>
<td></td>
<td>Caching of content and redirection of HTTP requests</td>
<td></td>
<td>Consistent hashing for mapping data to servers</td>
</tr>
<tr>
<td>Globule</td>
<td>Academic</td>
<td>Apache extension, Open Source collaborative CDN</td>
<td>Support for static content, monitoring services, DNS-based request-routing</td>
</tr>
<tr>
<td></td>
<td>Replication of content, server monitoring, redirection to available replicas</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Akamai

Clients fetch html document from primary server URLs for replicated content are replaced in html

Client resolves aXYZ.g.akamaitech.net hostname

Akamai.net name server returns NS record for g.akamaitech.net
G.akamaitech.net nameserver choses server in region

Should try to choose server that has file in cache - How to choose?

Uses aXYZ name and consistent hash
How Akamai Works

1. End-user requests "index.html" from cnn.com
2. cnn.com forwards request to DNS root server
3. DNS root server forwards request to Akamai server
4. Akamai server directs request to Akamai high-level DNS server
5. Akamai high-level DNS server forwards request to Akamai low-level DNS server
6. Akamai low-level DNS server forwards request to Akamai server
7. Akamai server forwards request to cnn.com
8. cnn.com forwards request to End-user's device
9. End-user device requests "foo.jpg"
10. cnn.com forwards request to DNS root server
11. DNS root server forwards request to Akamai server
12. Akamai server forwards request to cnn.com

Get index.html

Get/ cnn.com/foo.jpg

Source: www.cs.cmu.edu/~srini/15-744/S08/lectures/17-DNS.ppt
Coral: An Open CDN

Pool resources to dissipate flash crowds

Implement an open CDN
Allow anybody to contribute
Works with unmodified clients
CDN only fetches once from origin server

Runs in PlanetLab
Based on NSDI 2004 presentation and paper
Using CoralCDN

Rewrite URLs into “Coralized” URLs


Coral distributes the load

Who might “Coralize” URLs?

Web server operators Coralize URLs
Coralized URLs posted to portals, mailing lists
Users explicitly Coralize URLs
Basic DHT is not the solution

A Basic DHT, such as Chord, allows to:

- insert (URL, {IP1})
- lookup(URL) and get IP addresses

However, this construction does not:

- a single node is responsible for a specific URL
- no load balancing
- no proximity awareness
Coral

Coral addresses these limitations by:

Introducing proximity
   A way to find nearby servers based on IP prefixes

Having a hierarchical DHT organized based on latency

Supporting load balancing
Coral Server Discovery

1. Each Coral server inserts its IP **network prefix** as key, its IP address as value
2. DNS server does DHT lookup on client IP prefix to find nearby Coral server

Details (multiple prefixes):

Each Coral server uses traceroute to find nearby routers
Registers itself under IP of each nearby router

Coral DNS server traceroutes to client
Looks up each router IP address in mapping
DNS Redirection
Return proxy, preferably one near client
Hierarchical DHT

A hierarchy of DHTs, with clustering at lower levels
DHT based on XOR metric

Nearby (< 20 ms) Coral nodes form an L2 DHT
  L2: <20ms
  L1: 60 ms
  L0: global
Search in L2 DHT first
  If nearby copy exists, will find it first
Only search L1, L0 if miss in lower level
Finding URLs

Look up the URL in a DHT
key=URL, value=IP addr of Coral cache that has the URL

Coral cache fetches the page from that other cache

If DHT had more than one value for key, fetch page from more than one
In case one is down or slow
Coral reduces server load

Most hits in 20-ms Coral cluster

Local disk caches begin to handle most requests

Reduced traffic at origin

400+ PlanetLab nodes provide 32 Mbps, 100x capacity of origin

Source: www.news.cs.nyu.edu/~jinyang/fa07/notes/dht.ppt
Challenges for DNS Redirection

Coral lacks…

Central management

\textit{A priori} knowledge of network topology

Anybody can join system

Any special tools (e.g., BGP feeds)

Coral has…

Large number of vantage points to probe topology

Distributed index in which to store network hints

Each Coral node maps nearby networks to self
CDN Federation

Service Providers have implemented their own CDNs to avoid outsourcing this service
Service provider caches are closer to consumers

Service Providers have incentive to interconnect their CDN systems

CDN service sold by Service Providers becomes fragmented unless there is a way to interconnect and federate the different independent CDN systems

CDN Interconnection working group at the Internet Engineering Task Force (IETF)
While both models are distinct, it is possible that some aspects of their interworking may be performed through direct member interconnections while other aspects may be realized by interconnecting to a IETF exchange.

Cisco’s Open CDN Federation Pilot

To accelerate the development of IETF federations given the clear benefits of this approach, Cisco is working with several leading IETF and content providers worldwide to implement a long-term, large-scale, multi-phased initiative to plan, deploy, and test an open IETF federation pilot. The main goal is to move the IETF industry from a great idea to a market reality that can deliver clear benefits to IETF’s providers and offer a better experience for consumers while building a foundation for significant long-term network cost savings.

Cisco is collaborating with participating IETF’s and content providers in three areas:

- Business assessment: Identify the roles and responsibilities for each stakeholder as well as the associated and supporting business models and processes.
- Technical assessment: Assess the IETF interconnection architecture capabilities and approaches, as well as identify the technical roadmap for production deployment. This includes interconnection capabilities for request routing, content distribution, and accounting.
- Lab validation: Test and validate the concept of an open IETF federation through the worldwide interconnection of participating IETF’s and content sources in a lab environment.

Source: Cisco IBSG, 2011
Example of CDN Interconnection

http://en.wikipedia.org/wiki/Content_delivery_network_interconnection#mediaviewer/File:Cdni_usage_example.png
IETF Content Delivery Network Interconnection (CDNI) Interfaces
Key Technical Requirements

• Delivery of content simultaneously on multiple federated CDNs in arbitrary CDN topology meshing, including cascaded CDNs
• Hierarchical dynamic acquisition of content across CDNs
• Transparency to content provider
• Manually triggered inter-CDN content purge
• Single-vendor and dual-vendor operations
• Exchange of transaction logs in bilateral mode or via CDN Exchange
• Scalable provisioning
• Support for common HTTP-based VoD streaming, including adaptive streaming
• HTTP-based live adaptive streaming (Microsoft Smooth, Apple HLS, Adobe Zeri/HDS)
• Time-shifting video services
• Global as well as in-country/domestic federations
• Content with special delivery requirements
• Dense meshing with many-to-many simultaneous delivery
• Support for HTTP-based and DNS-based inter-CDN redirection
• Fine-grain access control policy via URI-signing
• High performance gain of CDN federation

P2P CDN

P2P techniques can be used to boost CDNs

CDN is the original source

P2P for local / regional delivery

Example: P4P (Sigcomm 2008)
YaCy distributed search engine based on a DHT

Source: http://yacy.net/en/index.html
Routing table is an AVL tree and elements are word hashes that point to hashed URL and hash of the peer that stores the full reference. Word entries are split among peers across 16 partitions of the address space.

http://lwn.net/Articles/469972/
http://www.yacy.net/en/Technology.html