15-441/641 Recitation

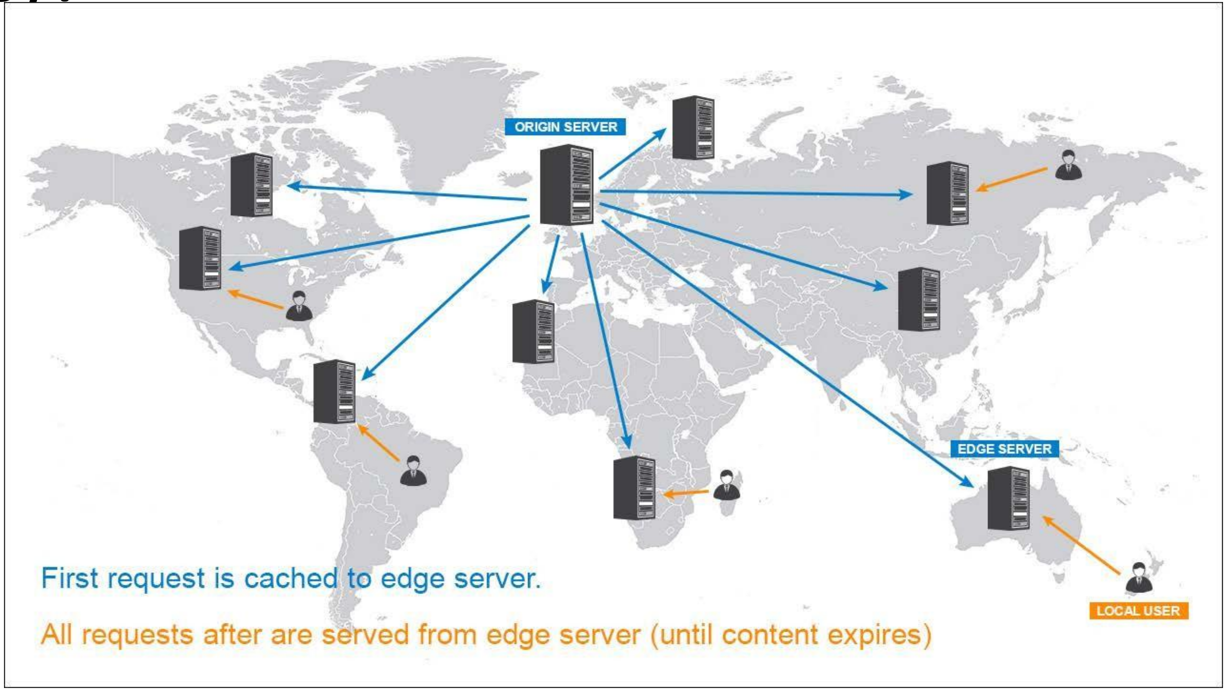
Project 3: Video CDN

Generations of TAs

Introduction to CDN

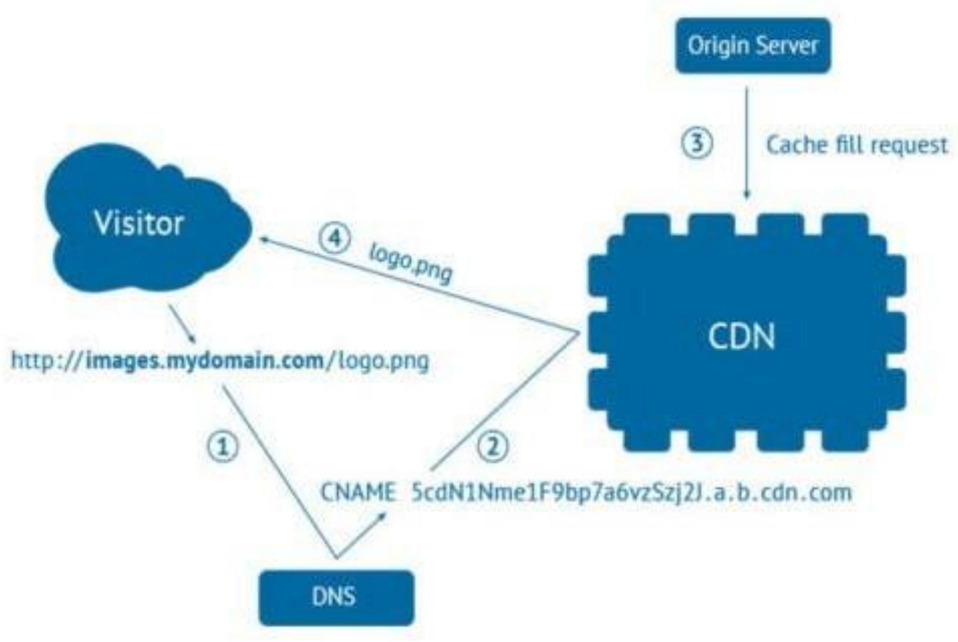
More details in Lecture 19 slides

What Are Content Delivery Networks (CDNs)?



Source:
https://howtogetonline.com/a-guide-to-content-delivery-networks.php

How Do CDNs Work?



Source:
https://www.technodoze.com/blogging/use-content-delivery-network.html

Why Use CDNs?

- Caching
  - Recall stuff from 213
  - CDN brings data closer to clients -> reduces propagation delay
- Reduce Load on One (or some) Servers
  - Central server hardware can be simplified
- Protects Against DDoS Attacks
  - Multiple points of failure

Actual CDN vs. Your CDN

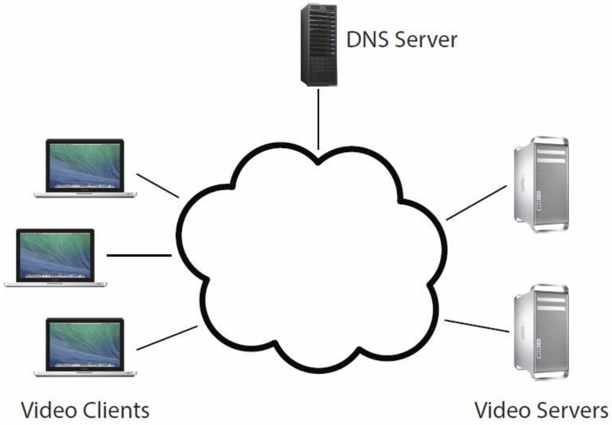


Figure 1: In the real world...

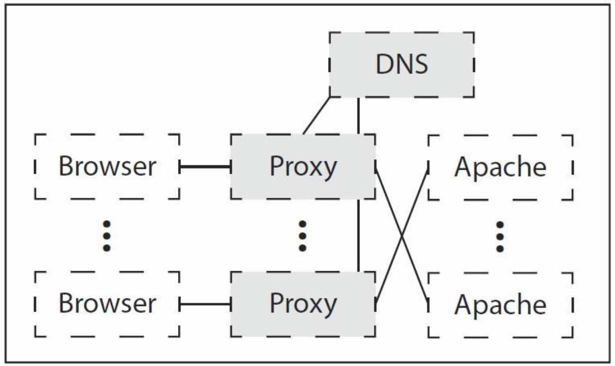


Figure 2: Your system.

Figure 3: System overview.

You Need to Implement

1. HTTP proxy with adaptive video bitrate selection
   - Intercept HTTP requests and modify them to request the bitrate appropriate for current network conditions.
   - Query the DNS server to determine which server to forward the request to.
2. DNS server with load balancing
   - Implements a subset of the DNS protocol.

Bitrate Adaptive Proxy

Proxy: TODO

1. Concurrent HTTP proxy basics from Project 1/Starter code
2. Calculate Throughput
   - *T_curr = Size_of_chunk*/(*time_end – time_start*)
   - *T_avg* ← α(*T_curr*) + (1- α)*T_avg*
3. Choose the bitrate of the chunk based on *T_avg*
   - The available bitrates can be found in the "*.f4m" file, requested at the beginning of the stream by the video player
   - NOTE: You won't send this file to the browser, instead you'll send a dummy file
4. Modify the request URL accordingly
   - /path/to/video/<*bitrate*>Seq<snum>-Frag<fnum>
5. Query the DNS server for the server IP using the interface in *mydns.h* **[checkpoint 2]**
6. Make the request

Running Your Proxy

proxy <log> <alpha> <listen-port> <fake-ip> <dns-ip> <dns-port> [<www-ip>]

- **Log**: Path to the log file [IMPORTANT: The grader relies on it]
- **Alpha**: A float in range [0,1]
- **Fake-ip**: You will bind to this IP address when you *connect* to the web-server
- **Dns-ip**: IP address of the DNS server
- **Dns-port**: UDP port on which the DNS server is listening
- **www-ip**: [OPTIONAL] The IP of the webserver. If not specified, the proxy should query the DNS server for the IP.

DNS with Load Balancing

DNS: TODO

1. Familiarize yourself with the format of a DNS message.
   - Refer to RFC 1035 and the writeup
2. Write constructors and parsers for DNS messages
3. Listen on a UDP socket for incoming DNS query packets
4. Implement round-robin load balancing
   - Just return the next server in the list
5. Implement shortest-path load balancing (next slide)
   6. Implement Dijkstra’s shortest path algorithm
   7. For each client determine the closest server using LSAs
6. Implement the interface in *mydns.h*

Shortest Path Load Balancing

1. Read LSAs from the specified file
   - Format: `<sender> <sequence number> <neighbors>`
     - Sender: IP address of the sender
     - Neighbors: comma-separated IP addresses of neighbors
2. Only consider the messages with the largest sequence number
3. Build a graph and run Dijkstra's algorithm

Running Your DNS Server

- nameserver [-r] <log> <ip> <port> <servers> <LSAs>
- **-r** [OPTIONAL] Uses round robin load balancing
- **Log**: log file [IMPORTANT: The grader relies on it]
- **Servers**: file containing the IP addresses of the servers
- **LSAs**: A file containing the LSAs, one per line

Implementing *mydns.h*

```
/**
 * Initialize your client DNS library with the IP address and port
   number
 * of your DNS server.
 * @param dns_ip The IP address of the DNS server.
 * @param dns_port The port number of the DNS server.
 * @return 0 on success, -1 otherwise
 */

    int init_mydns(const char *dns_ip, unsigned int dns_port){
      // Probably just initialize some internal data structures
    }
```

```
/**
 * Resolve a DNS name using your custom DNS server.
 * Whenever your proxy needs to open a connection to a web server, it
 * calls resolve() as follows:
 *   struct addrinfo *result;
 *   int rc = resolve("video.cs.cmu.edu", "8080", null, &result);  if (rc != 0) {//
 *   handle error }
 *   // connect to address in result
 *   free(result);
 * @param node The hostname to resolve.
 * @param service The desired port number as a string.
 * @param hints Should be null. resolve() ignores this parameter.
 * @param res The result. resolve() should allocate a struct addrinfo,
 * which the caller is responsible for freeing. * @return 0 on success, -1
   otherwise
 */

 int resolve(const char *node, const char *service, const struct addrinfo
*hints, struct       addrinfo **res){
   // Send the actual DNS request over UDP
}
```

Developing and Testing

Development Environment

- You will work on a preconfigured virtual machine
  - The virtual machine disk image can be downloaded from the assignments page as a “.vmdk” file
- You will have to setup a virtual machine with 64-bit Fedora as the operating system and import the disk image.
  - You may use any virtualization software for this but we recommend Oracle VirtualBox (it’s FREE!!)
- You will work on the “Project 3” account which has admin rights
  - Username: proj3
  - Password: project3
- You will find the starter code in the home directory

Network Simulation

- You will simulate the whole network, including proxies, routers and, web and DNS servers, on the virtual machine.
- You will use `netsim.py` for:
  - Simulating network topologies
    ```
    ./netsim.py <topology-dir> start/stop
    ```
  - Simulating events on the network
    ```
    ./netsim.py <topology-dir> run –r <events-file>
    ```
- A few topologies are included, but you are encouraged to make your own as well.

Topology Directory Structure

- topos/
  - <topology-name>/
    - <topology-name>.clients
      - List of IP addresses for the clients (proxies)
    - <topology-name>.servers
      - List of video servers
    - <topology-name>.dns
      - A single IP address for the DNS server
    - <topology-name>.links
      - A list of space-separated node pairs, e.g. 1.0.0.1 router1
    - <topology-name>.bottlenecks
      - List of links that you want to run events on
      - <ip1> link<num> <ip2>
    - *.events
      - <time> <link> <bandwidth> <latency>
    - *.lsa

Open the files and read the comments for more details