CDN Project

In this project, we will build a meta-CDN that optimizes content distribution for network and price performance. Please do this project in Python.

You should be able to obtain enough free server time on the major cloud vendors to do this project. You can do most testing on program.cs and your laptop.

Milestone 1 due: 10/31/2017
Milestone 2 due: 11/23/2017

Milestone 1:

1. Learn how to setup a Linux server on Amazon AWS, Microsoft Azure, and Google Cloud. For most of these assignemnts, lauching one server on each provider should be enough. For development and testing, using different port numbers and running them on program.cs should be ok.

2. Program a Python server "server.py" that you will run in the cloud and has two main capabilities:

   (1) accept a file upload from an external user. The user will upload the file using a custom Python program called "upload.py" that accepts the filename as command line argument. This program will use JSON to interact with the server. upload.py needs to upload any metadata along with the content. You will have to use proper encoding for the file content you will upload.

   (2) distribute the file to other servers. When one server receives a new upload, it should distribute the file to other servers to keep the content synchronized on all the servers. This also means each server is configured with a list of other servers. This is initially done using a configuration file when we launch the server. But the server should also provide an API to query and update the server list using a standard protocol. In summary, there is a standardized protocol for synchronization among the servers and a set of standardized API that provide administrative access to the servers. All messages use JSON.

3. Build a server (priceinfo.py) that provides price information for various service providers. You can run this server on one of the servers you instantiated. Provide the standardized API through which the users can query or update the network and server cost for serving content through different cloud service providers.

4. Write a HTTP proxy server proxy.py in Python. This proxy server should connect to one of the cloud servers, query that server for a list of known servers, and perform network measurements. The proxy server should also use the pricing API to obtain price information from the pricing server.

5. The proxy server runs on your client node and accepts a flag on the command line that determines if it will use purely network measurement, price, or a combination of the network measurement and price to decide the cloud server to which to route HTTP requests. Proxy will make this routing decision for each incoming HTTP request. Make the proxy multi-threaded so that it can process multiple requests at a time.

6. Create different network and pricing scenarios to trigger different routing decisions by proxy for the same request. You should be able

to test this proxy server and the whole setup using command line tools such as wget or curl or a browser. You can limit your tests to a simple document or a file.

Milestone 2:

1. Make sure your server can initialize its content from other servers if they are online. Your servers should send regular heartbeat messages to synchronize the list of "known" servers. This capability allows you to restart easily after a crash and bootstrap the content or to add new servers to increase your overall capacity. Make sure you perform a crash and restart test to make sure everything is working correctly.

2. Change the servers and upload.py so that they can switch between JSON encoding and Protobuf3 using a command line flag. Design experiements to decide which is better. Perform those experiments, analyze the results, including sanity checks, and write up the results.

3. Introduce secure (encrypted) communication between the servers when they synchronize the content. This functionality should also be enabled by a command line flag. You can use self-signed certificates or get a certificate from letsencrypt.org or try both.

4. Assume the content is images. Design a system that can automatically scale the content if the client is on a slow link. Implement two ways to do this content scaling. One approach is to "pre-compute" different versions of the same image at different resolutions and serve a version for particular data rate client. The other approach is to dynamically scale the image when a request for image comes in. How does the server know the data rate on the link? Create different network scenarios to thoroughly test this functionality and study the communication/computation overhead for differnet workloads.