PA5: Create your own content delivery network redirector

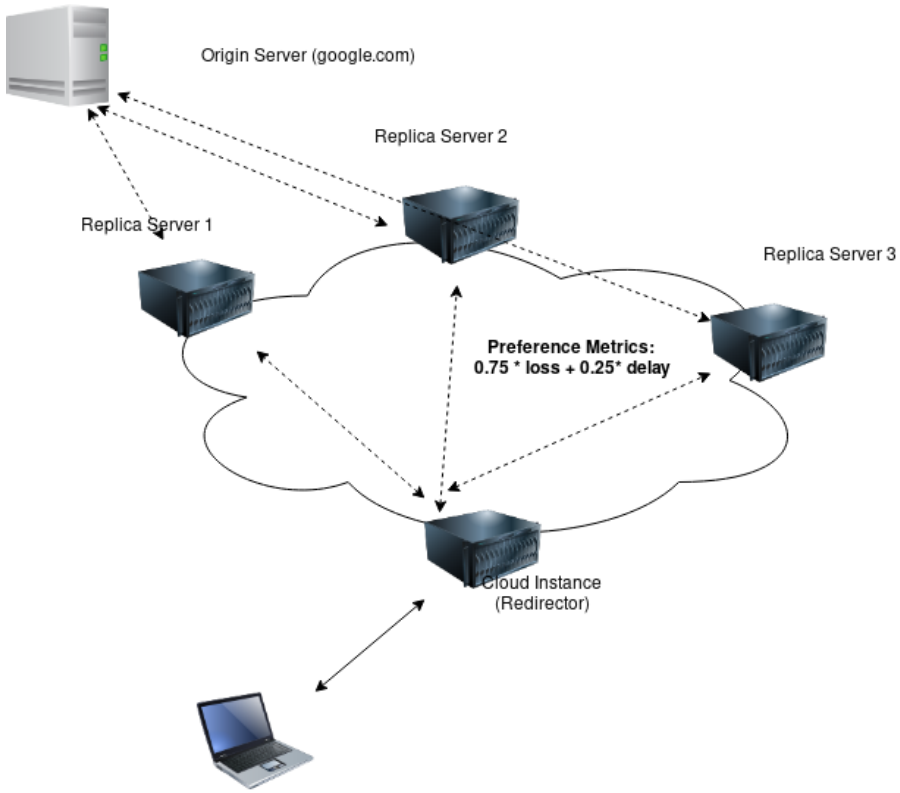
Due Date - April 26, 2022, 10PM CST

Objectives

---

1. Learn to break down a complex problem into smaller problems
2. Solve each smaller problem and bring them together to solve the larger problem
3. Learn how CDNs work and build your own CDN redirector

Overview



In this project, you are going to build on the previous projects. You will build your own "CDN load-balancer/redirector". A cloud typically hosts multiple copies (replicas) of the same content. Depending on where the request is coming from, it routes the requests a suitable replica. A primary component of any CDN is the entity called a load-balancer/redirector. The function of a load-balancer is route requests to the "most suitable" server. CDNs choose their own metric for defining which server is the most suitable. It can be the replica with the lowest latency, the replica with lowest loss, or a combination of multiple parameters.

In this exercise, you will create three replicas with the same content. You will also keep track of the delay and loss parameters to each replica. At the beginning, each replica will download some content from a website. When a request comes in, you will redirect the request to the most suitable replica (see the definition below). When the next request comes in, you will redirect that request to the most suitable replica at that time. The network conditions will change throughout the experiment.

Note that the load balancer does not act as a proxy. It simply tells the clients which server to connect to.

The client and server communications may use TCP or UDP.

Replica Server Specifications

The replica server takes three arguments:

$ replicaserver -p <PORT> -s <LOG FILE LOCATION> -w <web page to download>

1.PORT - The port server listens on.

2.Log file location - Where you will keep a record of actions.

3.w - Which webpage to download and serve.

For example:

$ replicaserver -p 30000 -l /tmp/logfile -w www.nytimes.com

Load Balancer Specifications

$ loadbalancer -s <SERVER-IP> -p <PORT> -l LOGFILE

The load balancer takes three arguments:

1.Server IPs - A list of replica servers' IP addresses

2.PORT - The port the servers listen on. They will all listen to the same port

3.Log file location - Where you will keep a record of packets you received.

For example:
$ loadbalancer -s replica_servers.txt -p 6543 -l LOGFILE

Client Specifications

The client (we will name it anonclient) takes four arguments:

$ anonclient -s <LOAD-BALANCER-IP> -p <PORT> -l LOGFILE -f <file_to_write_to>

The client takes three arguments:

1.LOAD-BALANCER-IP - The IP address of the load-balancer/redirector.

2.PORT - The port the server listens on.

3.Log file location - Where you will keep a record of packets you received.

4.File to write to - Where you will write the retrieved content.

For example:
$ anonclient -s 192.168.2.1 -p 6543 -l LOGFILE -f test.txt

Client Requirements

---

1. The client will simply open a connection to the load-balancer and request a content

2. Design your own protocol headers for most efficient communication between the client and the load-balancer
3. The client should be able to receive the content and write to a file
4. The client connects to the load balancer and receives the "best" IP and port from the list of replica servers
5. The client then connects to the replica server using the IP and port returned by the load balancer. It then downloads the file from the replica server.

Load Balancer Requirements

---

1. The load balancer pings the replica servers and observes the delay and bandwidth. It keeps a list of preference metrics using the delay and bandwidth.
2. The load balancer accepts connection from the client and returns the "best" replica server's IP and port.

Replica Server Requirements

---

1. The replica server starts up and downloads the content from the origin server.
2. It returns the content when requested by the client.
3. Design your own protocol headers for the communication between the client and the replica server.
4. The server should gracefully process the incorrect file name and log the error.

Functional requirements

---

1. You need to create more than one server that will be serving content.
2. The client needs to get the best server from the load balancer.
3. The client then needs to connect to this best server.

Protocol Specifications:

---

1. Design your own protocol headers for most efficient communication between the load-balancer and the replicas. You may reuse headers from PA1/PA2, or come up with your own header.
2. The client should connect to the replica server that has the highest preference.
3. We define the preference as lowest combined value of weighed delay and loss. Lower value wins.
4. **Preference = 0.75*loss percentage + 0.25*delay in milliseconds** - You will need to measure the loss and delay every 60 seconds. You may use ping.
5. The load-balancer should be able to accept and process multiple connection from clients at the same time.

Additional requirements:

---

1. Code must compile/run on Google Cloud Ubuntu VM (18.04) or later - we will test your code only on the VM.

2. For each packet received, log interaction at the load-balancer in the following format:

   Request from <CLIENT IP>. Redirecting to <Replica IP>, Preference <Preference>, Next Preference was <Next lowest preference> to <Replica IP>

3. If error occurs: log the following:

   "Error: Unable to handle request between <Client IP>, <Replica IP>, <Preference>, <Error details>

Set the loss and delay using these commands:

---

apt-get install -y kernel-modules-extra
apt-get install -y kernel-debug-modules-extra

4. reboot

Add loss or Delay

tc qdisc add dev <ethernet device, e.g, eth0> root netem delay 200ms
tc qdisc add dev <ethernet device, e.g, eth0> root netem loss 20%

Reset loss or delay

tc qdisc add dev <ethernet device, e.g, eth0> root netem delay <new delay>

Delete everything

tc qdisc del dev <ethernet device, e.g, eth0> root