Content Delivery Network

Cirstean Paul

No Institute Given

Abstract. The Internet is a huge network connected over ten of thousands of kilometers. Such distances creates latency in the network which may lead to longer response times. For this case, the content delivery network or CDN's were created. Their purpose is to shorten the distance between the clients on the servers in the system.
This project will consist in making a scaled-down replica of a CDN on TCP protocol. The messages will have a content alike the HTTP to keep the project more real.

1 Technologies chosen

1.1 Programming language

The app will try to be as close as possible to the hardware system. Because of this, the app will be written in C++. By doing so we have much more room the play around the project, because we can easily control multiple threads when needed.

1.2 App servers

The app will be splitten in 4 main servers. The first one is the Client. The client(external actor) will be the interface our users use to communicate with our system. It will just send messages to the Edge server and will be able to redirect the same request to a CDN node, when needed.

The "Edge server" is the entrance in our CDN infrastructure. This will be the main server which the client will call(in other words, we will have the DNS server name synced with the edge server ip address). The edge server will send a redirect link to the client to the most optimized cdn node for the specific client.

The "CDN node" will be a cdn server which will cache the responses from the origin server and will be able to respond to the client when called.

The "Origin server" is our source of truth. In reality, this will be a ftp server which is connected to the cdn infrastructure.

1.3 Protocols

All the servers will communicate with each other using TCP. The data transfer between the apps is crucial so we cannot risk the possibility of not receiving data by using UDP.

The format of the protocol in this app will be of form similar with the HTTP protocol : "status-code", "Content", .

2 Structure of app

* (in the Fig.1) represent requests which happen conditional of the state of the system.

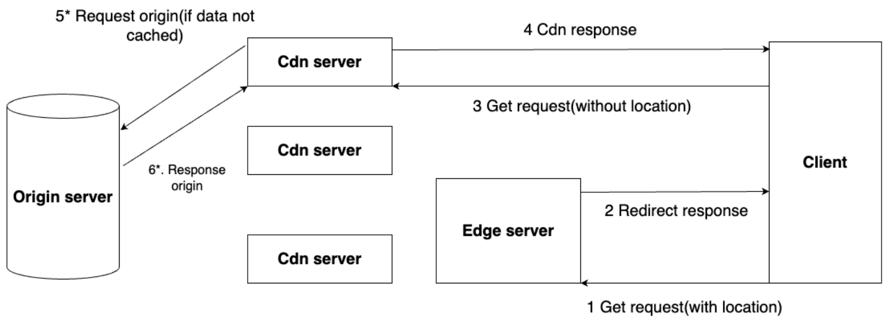

Fig. 1: General flow

The client will send a request to the edge server. This request will need a location attribute attached to it, so that the edge server may optimize which cdn node will be chosen for this client.

The client will need to get the redirect link and send the same request to the assigned cdn node(this time the request can have no location attached to it). The client will show the response of the cdn to the client. (More details on Fig. 2)

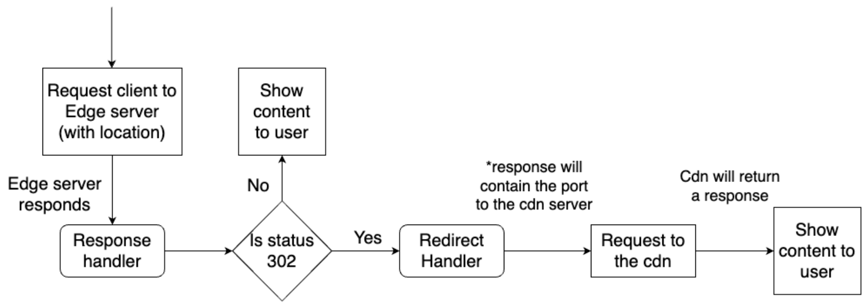

Fig. 2: Client to edge interaction

The edge server will respond to the client with a redirect link to the best CDN node for the specific client. (More details on Fig. 3)

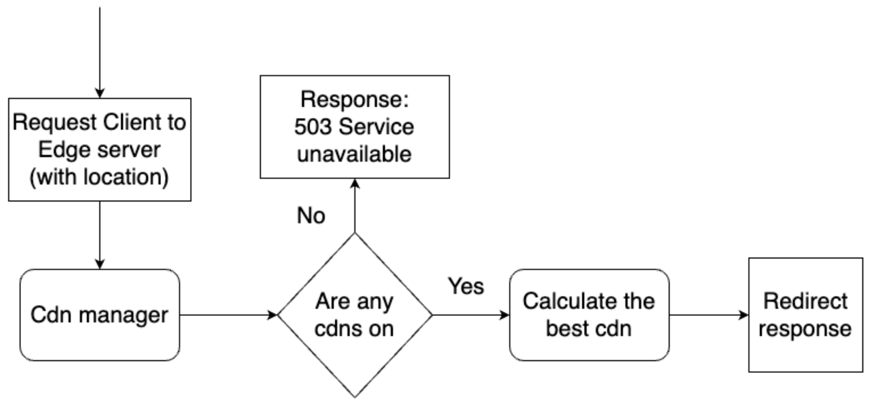
Request Client to Edge server (with location)
Response: 503 Service unavailable
No
Yes
Cdn manager
Are any cdns on
Calculate the best cdn
Redirect response

Fig. 3: Edge to client interaction

Also the edge server will get ”realtime” updates from all the cdns registered in the system. It will request these updates once in 3 seconds from each individual node. If it’s not able to connect to a cdn node, or the cdn node return a bad response, the edge server will consider that cdn node closed and will not consider it as a valid option to redirect a client to it. If the cdn return a good(200) response then the edge server will modify the load status of that cdn node. (Mode details on Fig. 4)

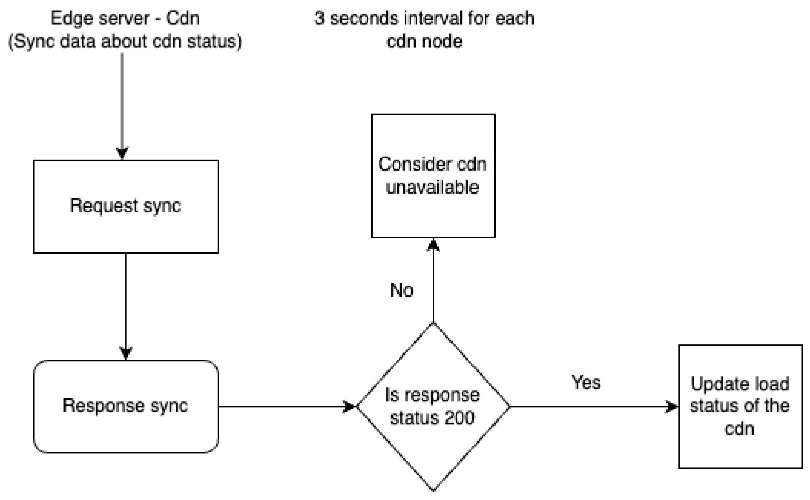
Edge server - Cdn (Sync data about cdn status)
3 seconds interval for each cdn node
Request sync
Consider cdn unavailable
No
Response sync
Is response status 200
Yes
Update load status of the cdn

Fig. 4: Edge to cdn interaction

After the redirect flow happens in the client, the client will send to the cdn the request. The assigned cdn will check the cache. It will try to find by the "prefix" of the request if the request is cacheable. If it is, then it will search in the cache that file. If the file is in the cache it will be returned to the client. If not, then the cdn will send the request to origin server and return it's content to the client. If the request is in status ok(200) then it's content will be added in the cache.

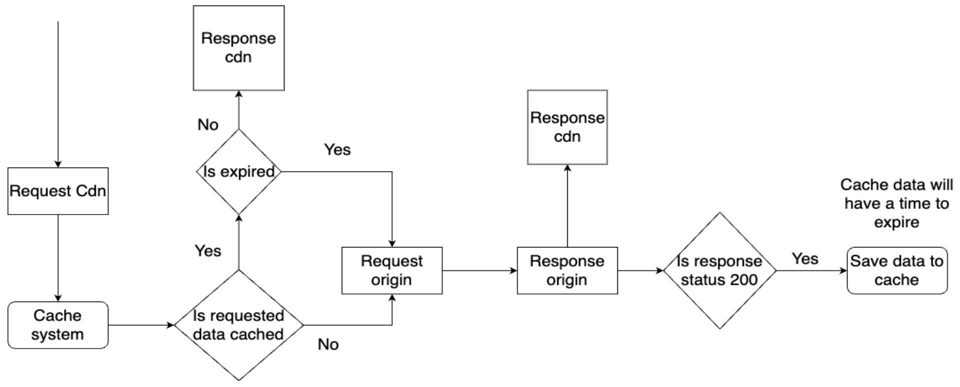

Fig. 5: Cdn to client interaction

The cdn is sending a request to the origin server. The origin server will verify the request(check if the command is known). If it's not then a bad request(400) will be returned. If the command is known the file specified in the request will be searched in the system. If it's not found then a not found(404) will be returned and if it was found then a ok(200) will be returned.

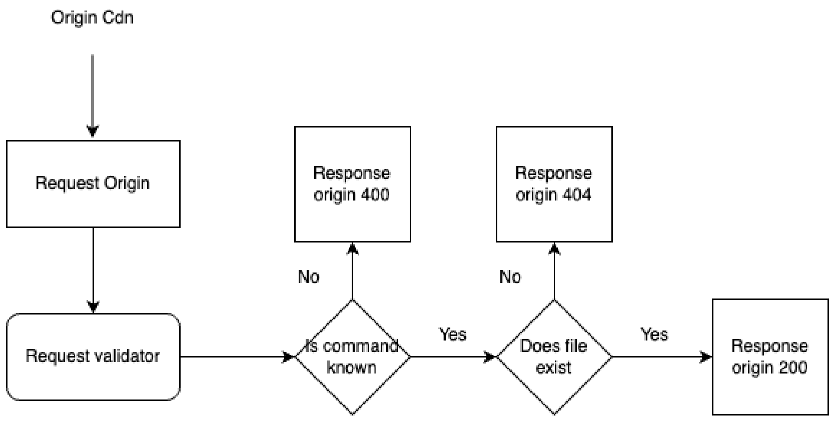

Fig. 6: Origin to Cdn interaction

3 Implementation remarks

The cdn node will need a separate channel to communicate with the edge server. This channel must be reliable, so the system is well optimized. Because there will be a data transfer between these 2 servers every 3 seconds, an open channel will be a logical solution to minimize the handshakes between the servers. So we will need a way to create something like a websocket. This channel must be open all the times, and if one end of the channel is connecting to it, the other end must know it. That being said, the solution found is to have the cdn server working as 2 servers on 2 different ports. One port will be assigned to deliver responses to the clients, and the other one will work just for the synchronization between the edge server and cdn node.

4 Improvements of the solution

The first down-side of the solution shown, is that there is no way to cache dynamic requests (such as API calls). A simple solution would be to add a flag in a request such as (Cache-control, in HTTP protocol).

Other important aspect is that the Edge-server is a Single point of failure. A solution possible in this scenario is to have a DNS server in which the fallback ips of the name server to be some cdn nodes.

References

1. Content Delivery Network. $https://en.wikipedia.org/wiki/Content_delivery_network$

2. Cloud-flare cdn system architecture. https://developers.cloudflare.com/reference-architecture/cdn-reference-architecture/
3. Http statuses https://developer.mozilla.org/en-US/docs/Web/HTTP/Status