Video Distribution Network

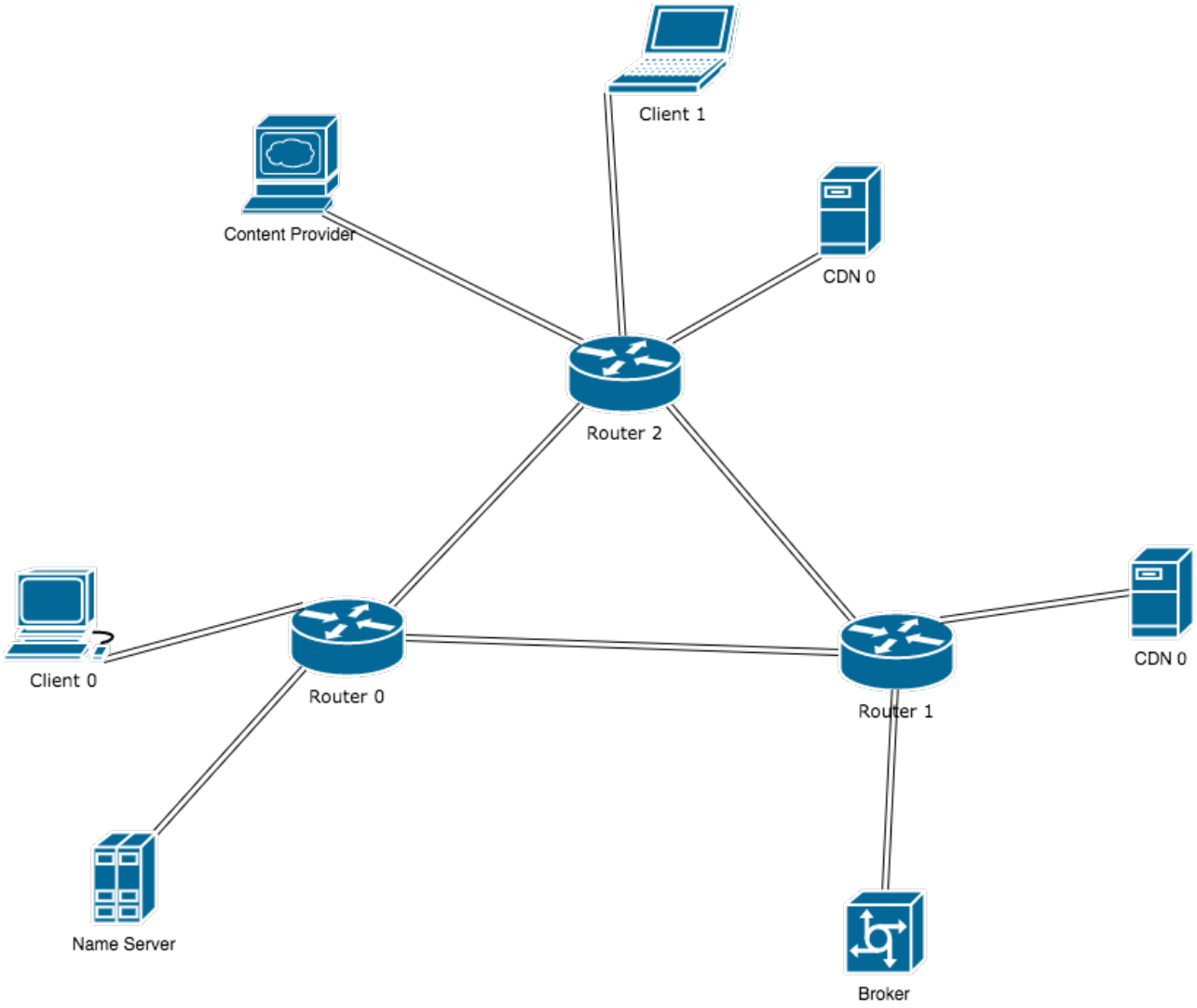
1. Overview

Traditional video distribution network gives client(the entity that wants to play the video) freedom to choose CDN by itself. The client always wants to maximize its throughput by choosing the best CDN it can use, and everyone within the network wants to achieve that, but no one cares about others. It has violated our purpose of maximizing user satisfaction(by saying satisfaction, we means increasing the resolution of the video, reducing latency, and decreasing buffering time) and minimizing cost at the same time if the client only cares about itself and competes for network resources with others. Always, it is necessary to sacrifice some lower-priority user (limit their throughput) to increase the overall network's satisfaction(the overall throughput and latency). It is obvious that if we let the client choose CDN, then no one will be willing to reduce the throughput to leave more network resources to others.

Therefore, we want to design a new video distribution network where the network resources are scheduled by the centralized network manager instead of each client, in that way, we can use some algorithm to make a scheduling plan that improves the entire network even if limits some clients.

2. Network Overview

We use XIA network as our network base. The following figure is the example of network topology:



Client 0, 1: The user that plays the video.

Name Server: The server that resolves the name of every device in the network.

Router 0, 1, 2: The device that routes all the network packet in XIA network.

Broker: The centralized network manager.

CDN 0, 1: The server where the user can request video chunks from.

Content Provider: The original video data provider, but in the network, it only serves as a manifest file server.

3. Process of Requesting Video

1. The client enters a web page that contains a video player with related video name(or URI).
2. Client clicks play icon on the player; then player first checks whether it has the manifest file of this video, if so go to step 6, otherwise go to step 3.
3. Before step 3, the player must know about the address of name server. Player checks whether it has cached the address of the manifest server(or content provider), if so go to step 5,

otherwise go to step 4.
4. Player asks name server for the address of content provider, and name server replies with the answer after that player caches the address associated with content provider's name.
5. Player requests manifest file from the content provider and caches the manifest file.
6. Player parses the manifest file, and get the list of CDNS that can be used to download the video.
7. Before step 7, the player must know the address of broker. Player set up a connection with the broker and tells broker it will going to download video from CDNS that are listed.
8. Broker record this request in the database, and prepared for scheduling CDN for the client.
9. Client downloads the video from CDN that is scheduled by the broker, and produces historical data such as throughput and RTT, and sends that data to the broker.
10. The broker receives the data, and perform a calculation on the network database to schedule the best CDN to the client.
11. Client keeps updating broker's scheduling and downloading videos.
12. Releases resources.

4. Setting up XIA network

Here we are going to use GENI test bed as our test environment.

4.1. SSH set up

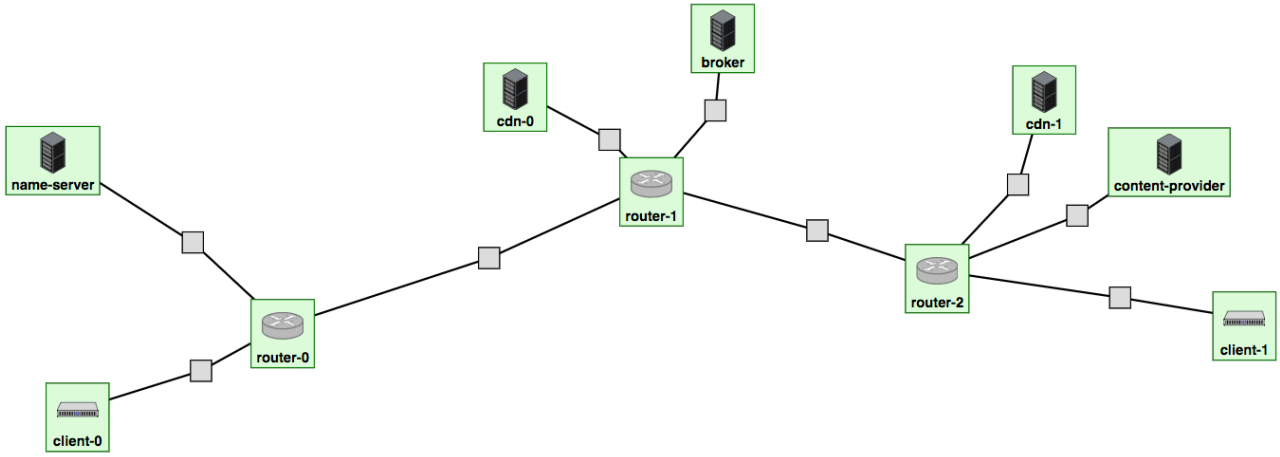
GENI uses SSH to verify user identity, so before we can set up network in GENI, we need first create SSH key pairs on our machine, and upload the public key to GENI, the guides of setting SSH is in SSH

4.2. GENI set up

1. Get permission from manager
2. Log into https://portal.geni.net/
3. Choose XIA project assigned by manager
4. Create a Slice by clicking "Create Slice"
5. Enter the name and the description as you want
6. Click create, and a new slice will be displayed
7. Click "Add Resources," an editable window will show up
8. Choose the box "Site 1" in the center, from the left sidebar, choose where you want your machine to be located.
9. Drag VM from left sidebar to the right, and then click the newly-dragged VM, modify its properties.
10. Change the name, and change "Disk Image" to "others". In the blank line, fill with "https://www.utahddc.geniracks.net/image_metadata.php?uuid=bd5b9b69-8655-11e6-9a79-000099989701"
11. Choose the corresponding icon according to the type of devices
12. To connect two devices, just drag the line from device A to device B

13. To change the bandwidth of the link, click on the link and enter the bandwidth number in the bandwidth blank
14. At the end of the page, click "Reserve Resources."
15. Wait for the result and do not close the "Waiting page" until the status is changed to "success", the reservation might take up to several minutes
16. If reservation fails, try another machine sites
17. On the Slice page, you will see the network topology that you create successfully, Choose "Detail", the status of each device will be listed. Also, all the VMs need some time to boot and get initialized after creation. This process will take several minutes
18. If the device icon in the Slice page turns green, then the device is ready to run.

The established network topology:



4.3. Connect to VM

In the "detail" page, there is a command that tells you how to SSH to each VM, for example:

Status	Client ID	Component ID	Expiration	Type	Hostname
READY	client-0	pc2	2017-12-09T23:59:59.000Z	emulab-xen	client-0.video.ch-geni-net.genirack.nyu.edu
Login	ssh barrettd@pc2.genirack.nyu.edu -p 30250 ssh qdeng@pc2.genirack.nyu.edu -p 30250				
Interfaces		**MAC**		**Layer 3**	
interface-0	pc2:eth2	02c40c814756		ipv4: 10.10.1.1	

Besides the command above, we should add "-i" argument to use SSH:

```
ssh -i ~/.ssh/id_geni_ssh_rsa qdeng@pc2.genirack.nyu.edu -p 30250
```

where "~/.ssh/id_geni_ssh_rsa" should be replaced by your SSH private key, after entering the command, you will be required to enter the password for the private key, which is the one you use when creating the SSH pairs.

4.4. Environment Set up

```
# get the latest library list
sudo apt-get update
# upgrade all of the library, this step must not be skipped, otherwise the GCC will report failure for unknown reason when compiling XIA
sudo apt-get upgrade
# install all the library that is required by XIA
sudo apt-get install git libxml2 libxml2-dev libssl-dev libprotobuf-dev protobuf-compiler python-protobuf python-networkx python-nacl python-pip python-tk python-dev python-requests python-crypto build-essential openssl swig
# download xia project
git clone https://github.com/XIA-Project/xia-core.git
cd xia-core
git checkout xia-v2
git pull
# configure make file
./configure
make
cd ..
# add XIA to path
sudo vim ./.bashrc
# [add export PATH=$PATH:~/xia-core/bin to file]
# eg:
# ...
# # Qiaoyu's import
# export PATH=$PATH:~/xia-core/bin
source ./.bashrc
# not required, you can enable SFTP if you want to transfer file between your machine and VM
sudo apt-get install openssh-server
sudo ufw allow 22
```

4.5. Database set up

Since broker will use the database to store all the historical data or even run OLTP query to pick CDN, we need install database and supporting development library:

On the VM that is used as the broker:

```
# 1. install postgresql
sudo apt-get install postgresql-9.6
# 2. configure user for the first time, ref:
# https://stackoverflow.com/questions/1471571/how-to-configure-postgresql-for-the-first-time
# 2.1. connect to the default database with user postgres
sudo -u postgres psql template1
# 2.2. set the password for user postgres, then exit psql (Ctrl-D)
ALTER USER postgres with encrypted password 'xxxxxxx';
# 2.3. edit the pg_hba.conf file
# and change "peer" to "md5" on the line concerning postgres
sudo vim /etc/postgresql/9.6/main/pg_hba.conf
# eg: local      all     postgres     md5
# 2.4. restart the database, use "psql -U postgres" to check
sudo /etc/init.d/postgresql restart
# 2.5. create a user having the same name as you (to find it, you can type whoami)
createuser -U postgres -d -e -E -l -P -r -s <my_name>
# 2.6. edit the pg_hba.conf file again
# and change "peer" to "md5" on the line concerning "all" other users
# local      all     all     md5
sudo vim /etc/postgresql/9.6/main/pg_hba.conf
# 2.7. restart again
sudo /etc/init.d/postgresql restart
# 3. now you can log in, create a new database that you will use to store data
createdb testdb
```

4.6. Configure XIA network

see the guides from XIA Wiki

Besides the guideline above for name server, use:

```
sudo bin/xianet start -r -n
```

for others, use:

```
sudo bin/xianet start -r
```

to test connection, use:

```
xping [hostname]
```

5. Run

Before running anything, we need download video file from dash1 or dash5, unzip them and create the directory structure like:

/resources +> /resources/dash1 +> /resources/dash5

5.1. Run XIA

On name server:

```
sudo bin/xianet start -r -n
```

On other VMs:

```
sudo bin/xianet start -r
```

And test connection:

```
xping [hostname]
```

5.2. Run CDNs

CDN needs do two things:

1. Register its name(e.g., cdn0, cdn1) to name server
2. Parse original manifest file and put the video chunk into XCache

This process is finished by video publisher:

1. On each CDN VM, edit the `.video-use-case/video_publisher.h`:

   change

   ```
   static const char* CDN_NAME = "www.cdn1.xia";
   ```

   to

   ```
   static const char* CDN_NAME = "www.[current CDN's name].xia";
   ```

   record "www.[current CDN's name].xia", and it will be used later when we generate manifest file on content provider.

2. Make the "video-use-case" project

3. run:

```
./video_publisher cdn
# video name should be dash1 or dash5 that is in the resources directory
# e.g.
# dash1
# cdn indicates that current machine will be used as a CDN,
# no need for generating manifest file, and the name we set above will
be
# registered to name server
[video name] cdn
```

   it will register "www.[current CDN's name].xia" to the name server, and put all the video chunk into Xcache

5.3. Run broker:

The broker needs to do two things:

1. Register its name(e.g., www_s.broker.xia) to name server
2. Listening on the socket for user request

This process is finished by broker server:

1. On broker, edit the `.video-use-case/broker_server.h`:

   change

```
#define BROKER_NAME "www.broker.xia"
```

   to

```
#define BROKER_NAME "www.[your broker's name].xia"
```

   record it, and every client must know it before the client starts running
2. Make the "video-use-case" project
3. run:

```
./broker_server
```

5.4. Run Content Provider

The content provider (or manifest server) needs to do three things:

1. Register its name(e.g., www.origin.xia) to name server.

2. Parse original manifest file to multi-cdn version.
3. Running manifest file server for sending client multi-cdn manifest file.

This process is finished by video publisher and manifest server:

1. On content provider, edit `.video-use-case/video_publisher.h`

   change

   ```
   static const char* MULTI_CDN_NAME = "www.origin.xia";
   ```

   to

   ```
   static const char* MULTI_CDN_NAME = "www.[your content provider's
   name].xia";
   ```

   Also, we need change

   ```
   static const char * MUTI_CDN_CDN_NAMES[] = {
       "www.cdn1.xia",
       "www.cdn2.xia",
   };
   ```

   to

   ```
   static const char * MUTI_CDN_CDN_NAMES[] = {
       "www.[cdn's name 1].xia",
       "www.[cdn's name 2].xia",
   };
   ```

   we need to ensure that all the CDN's names are included in this array, so content provider can generate the manifest file that client can use to route to CDN we built before.

2. On content provider, edit `.video-use-case/manifest_server.h`

   change

   ```
   #define VIDEO_ORIGIN_SERVER_NAME "www.origin.xia"
   ```

   to

   ```
   #define VIDEO_ORIGIN_SERVER_NAME "www.[your content provider's name
   above].xia"
   ```

3. Make the "video-use-case" project.

4. Run:

```
./video_publisher origin
# video name should be dash1 or dash5 that is in the resources directory
# e.g.
# dash1
# multicdn indicates that current machine will only be used to generate mutil-cdn
# version manfest file, without registering its name
[video name] multicdn
```

   It will parse the original manifest template into manifest file that can be used by client, which will be saved as `xia_manifest_dash1.mpd` in `video-use-case` folder, the way we parse will be discussed later.

5. Run:

```
./manifest_server
```

   to listen on the socket and be ready for sending manifest file back to client(this process needs xia_manifest_dash1.mpd has already been generated).

5.5. Run Client

Since dash it is hard to change its implementation, we use a proxy to implement the functionality that should have been put inside dash player. Therefore, the player knows nothing about XIA network, it only picks the chunk URL from manifest file, send HTTP request to proxy and expect video content is sent back via HTTP response. All the XIA and video distribution logic are hidden in the proxy.

In a word, the proxy is responsible for requesting manifest file with XIA API and communicating with the broker to choose different CDN to fetch chunks.

The client needs to do two things:

1. Run browser and redirect all the network flow to proxy
2. Run proxy in the background

Since we want to play video in client side, so we need SSH to client with x window mode, so first we need ensure that our machine has already installed "X11" or "XQuartz", so when we log into client VM, we need add `-Y` command after the SSH command:

```
ssh -i ~/.ssh/id_geni_ssh_rsa qdeng@pc2.genirack.nyu.edu -p 30250 -Y
```

And it will pop up an linux desktop window.

The process is:

1. Make the "video-use-case" project.
2. Run

   ```
   ./proxy 8080
   ```

   which means port 8080 will be listened by the proxy.
3. Start Firefox and configure HTTP proxy to 127.0.0.1 port 8080.
4. Go to URL:

   ```
   file:///[your-xia-core directory]/applications/video-use-case/src/visualizer/dash/samples/dash-if-reference-player/index.html
   ```

   And browser will open a dash player application inside the webpage.
5. Enter the URL in the address area of Dash player, and press Load:

   ```
   http://www.[your content provider's name above].xia/dash1.mpd
   ```

And then video begins to be played.

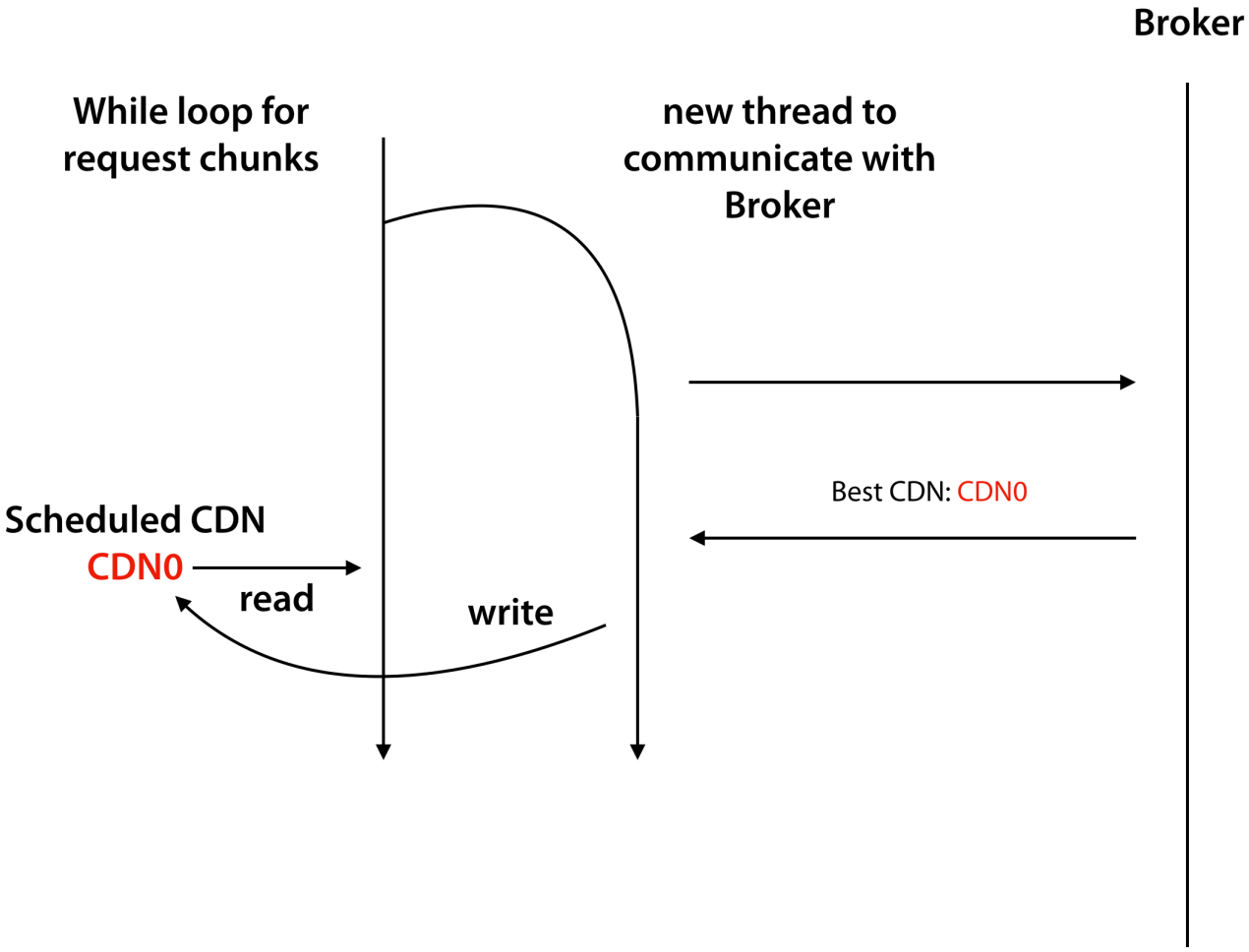
5. Broker Protocol

Before the broker is added into video distribution network, the client is response for choosing CDN and fetching video chunks, so it has very little delay between choosing and fetching. However, it is not the case when we add broker into network, the client has to send request to broker and get result back, the delay is one round trip time, which cannot be ignored in real network. Therefore, we must develop a protocol between client and broker that can hides the latency.

Therefore, we decide to divide the CDN choosing part and chunk fetching part into two separated threads, they can execute at the same time. Video fetching process will never be block even if no CDN result has been returned by broker, in that case, the client will choose a CDN randomly from the available CDN lists. It is asynchronous for the communication between client and broker, but the network status can reflect the future in most of time.

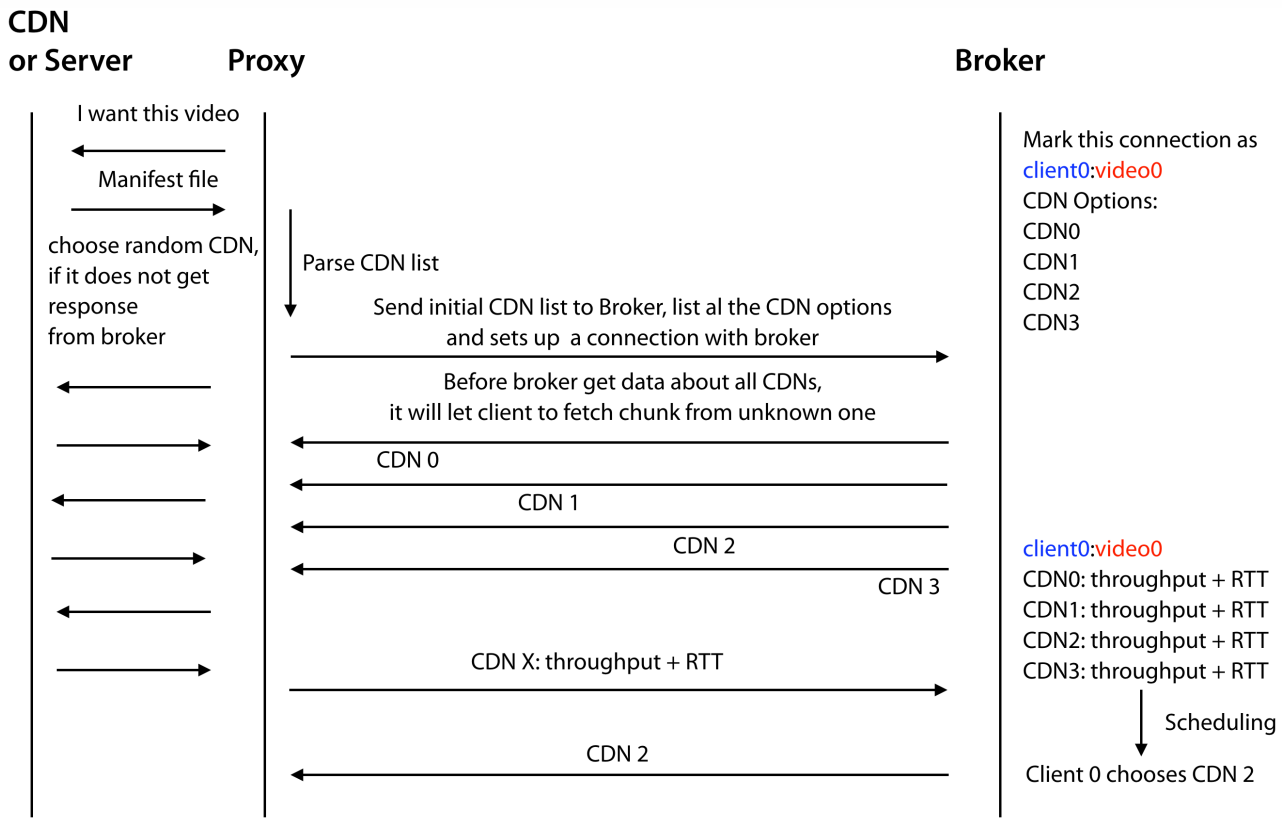
5.1. Protocol Model

The protocol model is:



5.2. Workflow

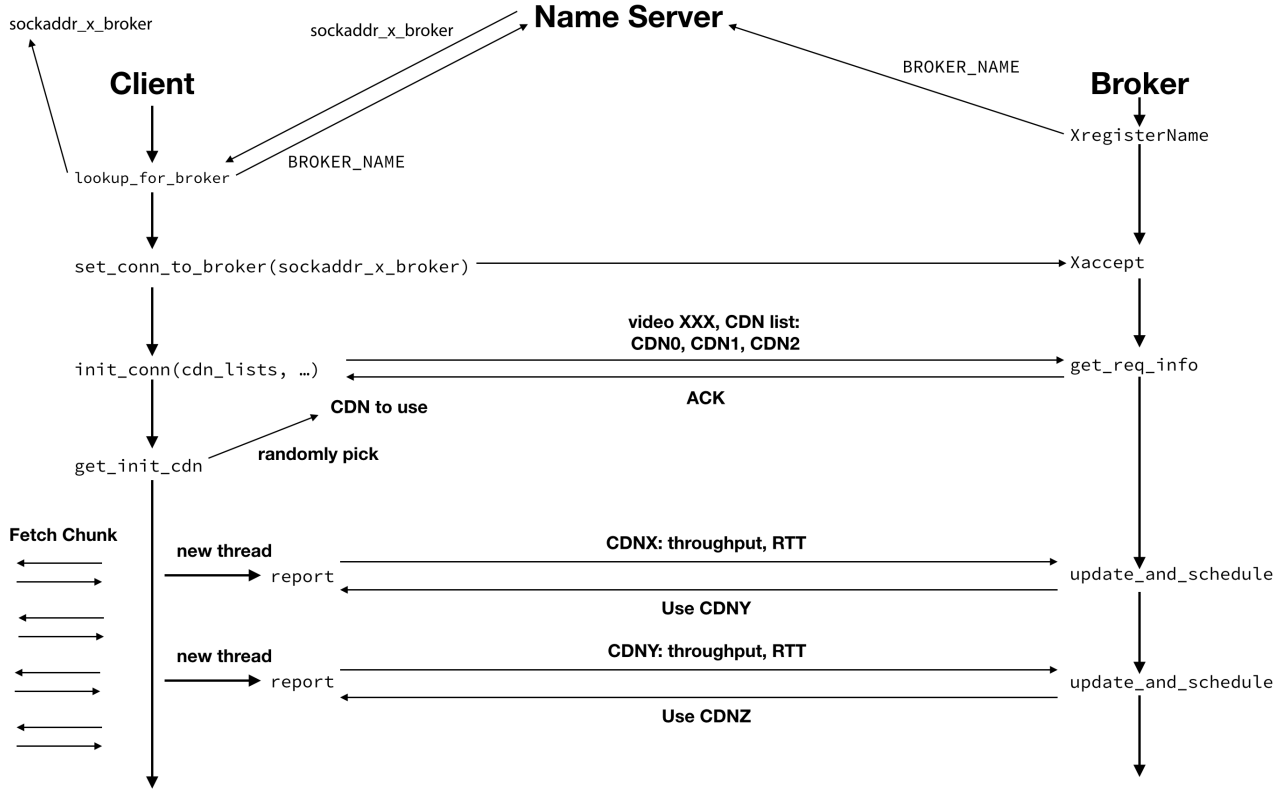
The workflow among client, broker, CDN should be:



5.3. Interface

Detailed interface documentation can be found in source file:

```
broker_client.h
broker_server.h
```



6. Manifest File

Since the original manifest file gets from Dash is only a template, we need parse it and generate manifest for every video chunk, to make it easier for dash player to download video chunk via HTTP request. Therefore, we parse and reformat the manifest file in the following two steps:

1. template-base manifest to chunk-base manifest
2. chunk-base manifest to chunk-base manifest with CDN information

the original video file structure is:

```
<MPD >
    <Period>
        <AdaptationSet mimeType="video/mp4">
            <Representation id="480_1200000">
                <SegmentTemplate
media="480_1200000/dash/segment_$Number$.m4s"/>
            </Representation>
            <Representation id="480_800000">
                <SegmentTemplate
media="480_800000/dash/segment_$Number$.m4s"/>
            </Representation>
            <Representation id="480_400000">
                <SegmentTemplate
media="480_400000/dash/segment_$Number$.m4s"/>
            </Representation>
        </AdaptationSet>
    </Period>
</MPD>

```

After we parse it to chunk-base manifest(expand it from bitrate template to chunk URL), it will become:

```
<MPD >
    <Period>
        <AdaptationSet mimeType="video/mp4">
            <Representation id="480_1200000">
                <SegmentList duration="114750" timescale="30000">
                    <Initialization sourceURL="480_1200000/dash/init.mp4"/>
                    <SegmentURL media="480_1200000/dash/segment_0.m4s"/>
                    <SegmentURL media="480_1200000/dash/segment_1.m4s"/>
                    <SegmentURL media="480_1200000/dash/segment_2.m4s"/>
                    <SegmentURL media="480_1200000/dash/segment_3.m4s"/>
                    ...
                    <SegmentURL media="480_1200000/dash/out.mp4"/>
                </SegmentList>
            </Representation>
            <Representation id="480_800000">
                <SegmentList duration="114750" timescale="30000">
                    <Initialization sourceURL="480_800000/dash/init.mp4"/>
                    <SegmentURL media="480_800000/dash/segment_0.m4s"/>
                    <SegmentURL media="480_800000/dash/segment_1.m4s"/>
                    <SegmentURL media="480_800000/dash/segment_2.m4s"/>
                    <SegmentURL media="480_800000/dash/segment_3.m4s"/>
                    ...
                    <SegmentURL media="480_800000/dash/out.mp4"/>
                </SegmentList>
            </Representation>
            <Representation id="480_400000">
                <SegmentList duration="114750" timescale="30000">
                    <Initialization sourceURL="480_400000/dash/init.mp4"/>
                    <SegmentURL media="480_400000/dash/segment_0.m4s"/>
                    <SegmentURL media="480_400000/dash/segment_1.m4s"/>
                    <SegmentURL media="480_400000/dash/segment_2.m4s"/>
                    <SegmentURL media="480_400000/dash/segment_3.m4s"/>
                    ...
                    <SegmentURL media="480_400000/dash/out.mp4"/>
                </SegmentList>
            </Representation>
        </AdaptationSet>
    </Period>
</MPD>

```

And the next step is to change the media field of each video chunk to HTTP URL that can be used by Dash player to fetch chunks, which is:

```
<MPD >
    <Period>
        <AdaptationSet mimeType="video/mp4">
            <Representation id="480_1200000">
                <SegmentList duration="114750" timescale="30000">
                    <Initialization sourceURL="http://[src]/[CID]?[CDN0]&
[CDN1]"/>
                    <SegmentURL media="http://[src]/[CID]?[CDN0]&[CDN1]"/>
                    <SegmentURL media="http://[src]/[CID]?[CDN0]&[CDN1]"/>
                    <SegmentURL media="http://[src]/[CID]?[CDN0]&[CDN1]"/>
                    <SegmentURL media="http://[src]/[CID]?[CDN0]&[CDN1]"/>
                    ...
                    <SegmentURL media="http://[src]/[CID]?[CDN0]&[CDN1]"/>
                </SegmentList>
            </Representation>
            <Representation id="480_800000">
                <SegmentList duration="114750" timescale="30000">
                    <Initialization sourceURL="http://[src]/[CID]?[CDN0]&
[CDN1]"/>
                    <SegmentURL media="http://[src]/[CID]?[CDN0]&[CDN1]"/>
                    <SegmentURL media="http://[src]/[CID]?[CDN0]&[CDN1]"/>
                    <SegmentURL media="http://[src]/[CID]?[CDN0]&[CDN1]"/>
                    <SegmentURL media="http://[src]/[CID]?[CDN0]&[CDN1]"/>
                    ...
                    <SegmentURL media="http://[src]/[CID]?[CDN0]&[CDN1]"/>
                </SegmentList>
            </Representation>
            <Representation id="480_400000">
                <SegmentList duration="114750" timescale="30000">
                    <Initialization sourceURL="http://[src]/[CID]?[CDN0]&
[CDN1]"/>
                    <SegmentURL media="http://[src]/[CID]?[CDN0]&[CDN1]"/>
                    <SegmentURL media="http://[src]/[CID]?[CDN0]&[CDN1]"/>
                    <SegmentURL media="http://[src]/[CID]?[CDN0]&[CDN1]"/>
                    <SegmentURL media="http://[src]/[CID]?[CDN0]&[CDN1]"/>
                    ...
                    <SegmentURL media="http://[src]/[CID]?[CDN0]&[CDN1]"/>
                </SegmentList>
            </Representation>
        </AdaptationSet>
    </Period>
</MPD>
```

So the dash player will generate HTTP request in the format of `http://[src]/[CID]?[CDN0]&[CDN1]`, and we can parse the URL in the proxy side to fetch video chunk in XIA network.

7. Issue

The communication protocol between the broker and client is non-stateless, which means both the broker and client needs to keep the current request alive, and always use the same connection to talk. However, the dash player is stateless because it treats each chunk as a separated resource, and it only follows the media URL to download chunks, with the help of proxy. Thus, the proxy needs a mechanism to identify which video request this chunk belongs to, and we map video request to a socket connected with the broker.

To achieve that, the proxy will add one more field in the manifest file, `REQUEST_ID`, when the client sends the initial video request to get the manifest. And the manifest file that is sent to the client will be:

```
1  <SegmentURL media="http://[REQUEST_ID]@[src]/[CID]?[CDN0]&[CDN1]"/>
```

In this way, even if the client knows nothing about current connection, the proxy can identify the video request easily by looking into the HTTP request, and always use the same connection with the broker to report current network status.

However, there is another problem with this method. The proxy will never know when to release the connection with the broker for the current video request because Dash player assume that all the network resources will be freed after fetching each chunk, and this is not true, we have not figured it out how to fix this resource leak problem. Here is some thinking about how to resolve it:

1. Change the implementation of dash player, let it send a "connection close" HTTP request after it downloads all the chunks, and the proxy will close release the resources once it receives "connection close".
2. If we assume that the throughput between client and CDN is the same for all the video requests, then we can change the broker protocol to stateless, and there is no need for identifying which video request this CDN data belongs to.
3. Change the dash player to operate on the multi-CDN manifest file directly, without proxy.
4. Set timeout for each request, if the request has not been called for a fixed amount o time, then the proxy tries to release it.