Proposta da reunião passada:

Avaliar o artigo: “Performance evaluation of video streaming using MPEG DASH, RTSP, and RTMP in mobile networks”. E simular testes parecidos com diferentes protocolos, sendo eles MPEG DASH, RTMP, HLS e HTTP e analisar seus resultados tendo como base os parâmetros de avaliação da qualidade de streaming, de acordo com algumas normas do ITU-T (International Telecommunication Union) Rec P.1201.1 (Series P: Telephone Transmission Quality, Telephone Installations, Local Line Networs):

- Velocidade de transmissão de vídeo
- Delay
- Resolução do vídeo
- etc.

Conhecendo o protocolo RTMP

Formato da mensagem

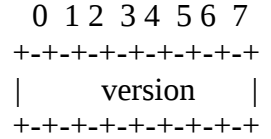
The message format SHOULD however contain the following fields:

- Timestamp: Timestamp of the message. This field can transport 4 bytes.
- Length: Length of the message payload. 3 bytes in chunk header. Se o cabeçalho da mensagem não puder ser omitida, então deve-se inserir no tamanho.
- Type Id: A range of type IDs are reserved for protocol control messages. This field occupies 1 byte in the chunk header.
- Message Stream ID: The message stream ID can be any arbitrary value. This field occupies 4 bytes in the chunk header in little endian format.

Handshake

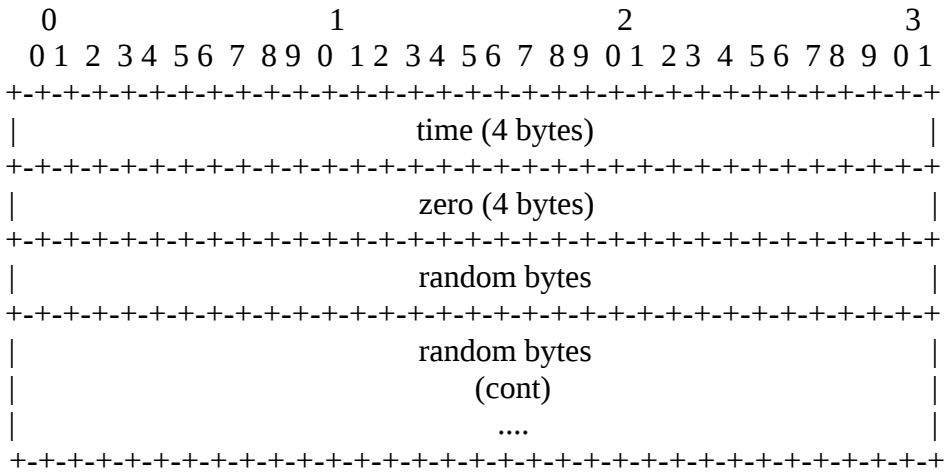
The client (the endpoint that has initiated the connection) and the server each send the same three chunks. For exposition, these chunks will be designated C0, C1, and C2 when sent by the client; S0, S1, and S2 when sent by the server.

C0 and S0 Format



Version (8 bits): In C0, this field identifies the RTMP version requested by the client. Values 0-2 are deprecated values used by earlier proprietary products; 4-31 are reserved for future implementations; and 32-255 are not allowed (to allow distinguishing RTMP from text-based protocols, which always start with a printable character). A server that does not recognize the client’s requested version SHOULD respond with 3.

C1 and S1 Format



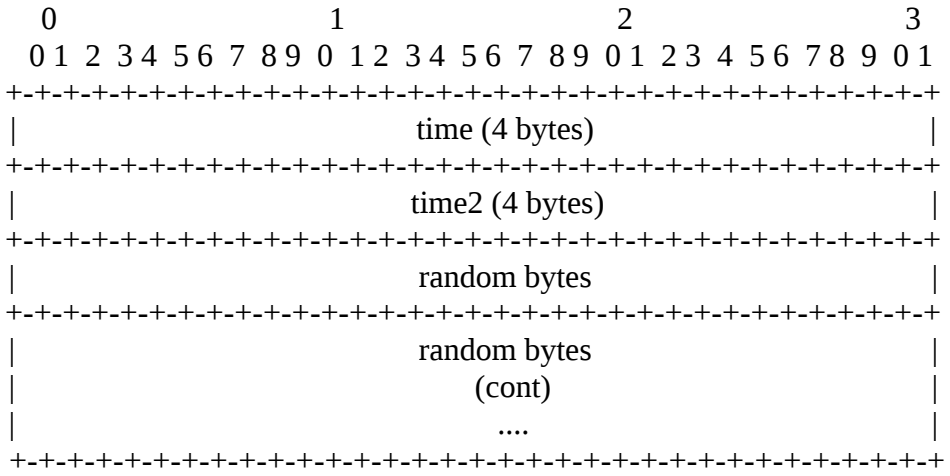
The C1 and S1 packets are 1536 octets long.

Time (4 bytes): This field contains a timestamp, which SHOULD be used as the epoch for all future chunks sent from this endpoint. This may be 0, or some arbitrary value. To synchronize multiple chunkstreams, the endpoint may wish to send the current value of the other chunkstream's timestamp.

Zero (4 bytes): This field MUST be all 0s.

Random data (1528 bytes): This field can contain any arbitrary values. Since each endpoint has to distinguish between the response to the handshake it has initiated and the handshake initiated by its peer,this data SHOULD send something sufficiently random.

C2 and S2 Format

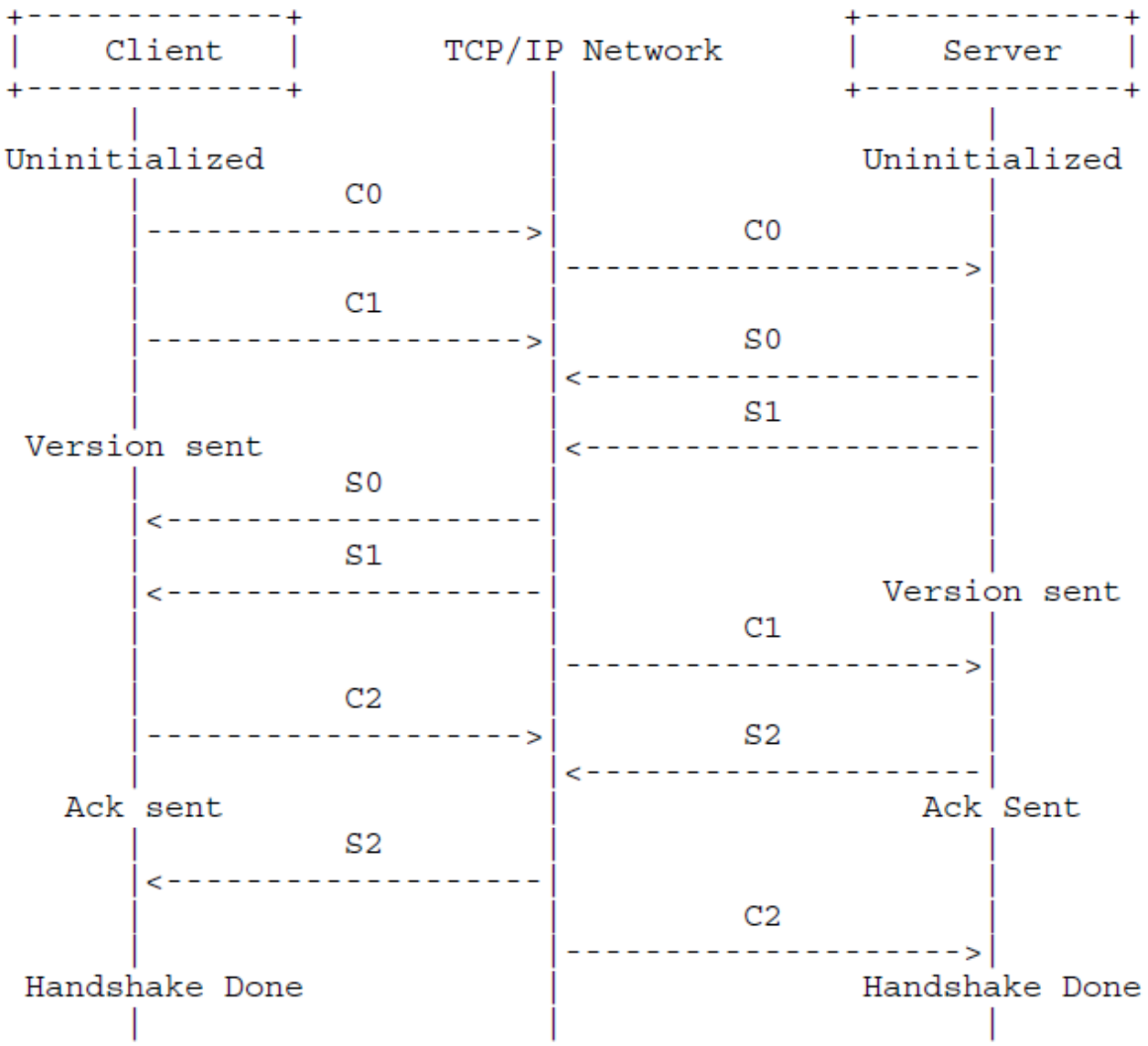


Time (4 bytes): This field MUST contain the timestamp sent by the peer in S1 (for C2) or C1 (for S2).

Time2 (4 bytes): This field MUST contain the timestamp at which the previous packet(s1 or c1) sent by the peer was read.

Random echo (1528 bytes): This field MUST contain the random data field sent by the peer in S1 (for C2) or S2 (for C1).

The C2 and S2 packets are 1536 octets long.



Pictorial Representation of Handshake

Uninitialized: The protocol version is sent during this stage. Both the client and server are uninitialized.

- The client sends the protocol version in packet C0.
- If the server supports the version, it sends S0 and S1 in response.
- If not, the server terminate the connection.

Version Sent: Both client and server are in the Version Sent state after the Uninitialized state.
- The client is waiting for the packet S1 and the server is waiting for the packet C1.
- On receiving the awaited packets, the client sends the packet C2 and the server sends the packet S2.

Ack Sent: The client and the server wait for S2 and C2 respectively.
Handshake Done: The client and the server exchange messages.

Chunking

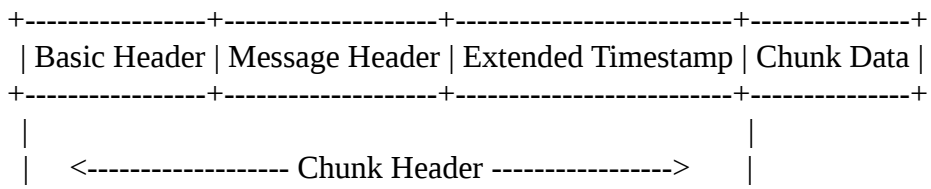
After handshaking, the connection multiplexes one or more chunk streams. Each chunk stream carries messages of one type from one message stream. Each chunk that is created has a unique ID associated with it called chunk stream ID.

Chunking allows large messages at the higher-level protocol to be broken into smaller messages.

The chunk size is configurable.The chunk size is configurable.

Larger chunk sizes reduce CPU usage, but also commit to larger writes that can delay other content on lower bandwidth connections. Smaller chunks are not good for high bit rate streaming.

Chunk Format



Each chunk consists of a header and data.

Conhecimento da ferramenta

Para começar a avaliação, foi necessária a instalação da ferramenta Nginx, que, segundo o próprio site da ferramenta o descreve como: “NGINX is a free, open-source, high-performance HTTP server and reverse proxy, as well as an IMAP/POP3 proxy server”.

Será utilizada a versão 1.10.3, que é a versão estável mais atual hoje.
Para ser instalada, foi necessário fazer alguns downloads, como por exemplo o da própria ferramenta, através do github, como também um módulo de RTMP, utilizado para realizar streamings RTMP, HLS, MPEG DASH e HTTP.

O download está disponível em [4].

Configuração do servidor

Após feito o download do arquivo compactado da ferramenta, é necessário carregar as configurações do servidor, para isso foi utilizado o comando:

```
$./configure --add-module=/root/nginx/nginx-rtmp-module/ --with-http_ssl_module
```

Atribuindo o módulo rtmp baixado ao arquivo de configuração.

Após realizada a configuração de alguns aquivos foi necessário executar o comando:

```
$make
```

E após, o comando:

```
$make install
```

Fazendo assim, a compilação dos códigos do Nginx.

Com os arquivos configurados, foi-se necessário alterar as configurações iniciais do servidor. Para isso, utilizou-se o arquivo de configuração *nginx.conf* gerado em:

```
/usr/local/nginx-streaming/conf
```

Colocando o seguinte código:

```
worker_processes  1;

error_log  /usr/local/nginx/logs/error.log;

events {
      worker_connections  1024;
}

http {
      include mime.types;
      default_type application/octet-stream;

      access_log /usr/local/nginx/logs/access.log;
      access_log /usr/local/nginx/logs/application_access_log.log;

      sendfile    on;

      keepalive_timeout  65;

      server {
            listen           127.0.0.1;
            server_name      10.0.0.108;

            # rtmp stat
            location /stat {
                  rtmp_stat all;
                  rtmp_stat_stylesheet stat.xsl;
            }

            location /stat.xsl {
                  # you can move stat.xsl to a different location
                  root /usr/local/nginx/html;
            }
```

```
            # rtmp control
            location /control {
                  rtmp_control all;
            }

            location / {
                  root   html;
                  index  index.html index.htm;
            }

            error_page 500 502 503 504  /50x.html;

            location = /50x.html {
                  root html;
            }
      }
}

rtmp {

      server {
            listen 1935;
            chunk_size 4096;

            application live {
                        live on;
                        record off;
            }

            # video on demand for flv files
            application vod{
                  play /home/nothereboy/Documents/NginxServer/flvs;
            }

            # video on demand for mp4 files
            application vod2{
                  play /home/nothereboy/Documents/NginxServer/mp4s;
            }

            access_log /usr/local/nginx/logs/access.log;
            access_log /usr/local/nginx/logs/application_access_log.log;
      }
}
```

Em que cada linha será explicada posteriormente. Criando assim um servidor RTMP *On Demand* para streaming de vídeo, em que os arquivos de streaming ficarão armazenados no seguinte diretório:

```
/home/nothereboy/Documents/NginxServer
```

Em que, dentro da pasta NginxServer, por enquanto, terão a pasta `flvs` onde ficarão os vídeos em formato .flv (Flash Video) e a pasta `mp4s` onde ficarão os vídeos em formato .mp4 (MPEG-4).
Outra informação importante a ser analisada são onde ficarão os arquivos e pastas de configuração do servidor Nginx, importante pois, dependendo do método de instalação, os arquivos podem ser alocados para a pasta etc, desconfigurando assim o servidor. Sendo assim, os locais a seguir são os caminhos de

cada arquivo:

nginx path prefix:	"/usr/local/nginx"
nginx binary file:	"/usr/local/nginx/sbin/nginx"
nginx modules path:	"/usr/local/nginx/modules"
nginx configuration prefix:	"/usr/local/nginx/conf"
nginx configuration file:	"/usr/local/nginx/conf/nginx.conf"
nginx pid file:	"/usr/local/nginx/logs/nginx.pid"
nginx error log file:	"/usr/local/nginx/logs/error.log"
nginx http access log file:	"/usr/local/nginx/logs/access.log"
nginx http client request body temporary files:	"client_body_temp"
nginx http proxy temporary files:	"proxy_temp"
nginx http fastcgi temporary files:	"fastcgi_temp"
nginx http uwsgi temporary files:	"uwsgi_temp"
nginx http scgi temporary files:	"scgi_temp"

Inicialização do Servidor

Para que o servidor comece a rodar, foi-se necessário executar o comando:

```
$sudo /usr/local/nginx/sbin/nginx
```

E caso quisesse que o servidor parasse de funcionar, seria necessário utilizar o comando

```
$sudo /usr/local/nginx/sbin/nginx -s stop
```

Teste de streaming

Para realizar os testes de streaming, foi necessário utilizar o rtmpdump para baixar o arquivo de dados

```
$sudo rtmpdump -r rtmp://192.168.0.16:1935/vod2/sample2.mp4 -o sample2.mp4
```

Para teste de veracidade também foi usado o programa VLC com o duas streams, sendo os links:

```
rtmp://192.168.0.16:1935/vod2/sample1.mp4
rtmp://192.168.0.16:1935/vod2/sample2.mp4
```

Resultado do streaming

Foram realizadas 4 transmissões de vídeo, sendo as características de cada vídeo:

Video: sample0.mp4
Tamanho: 383.6 kB (383,631 bytes)

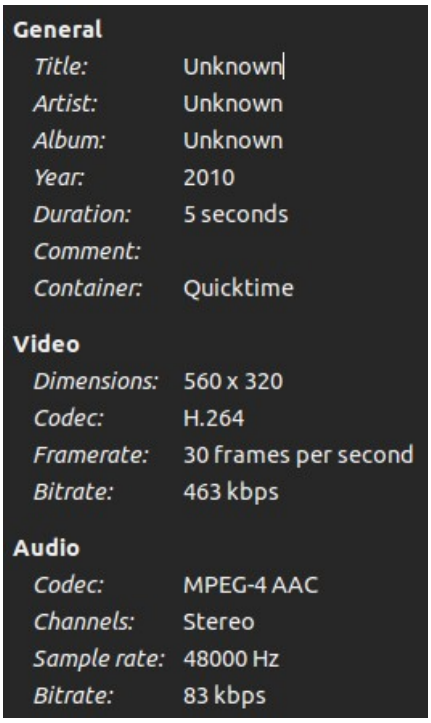


Imagem 1: Características do sample0.mp4

Video: sample1.mp4
Tamanho: 31.7 MB (31,661,978 bytes)

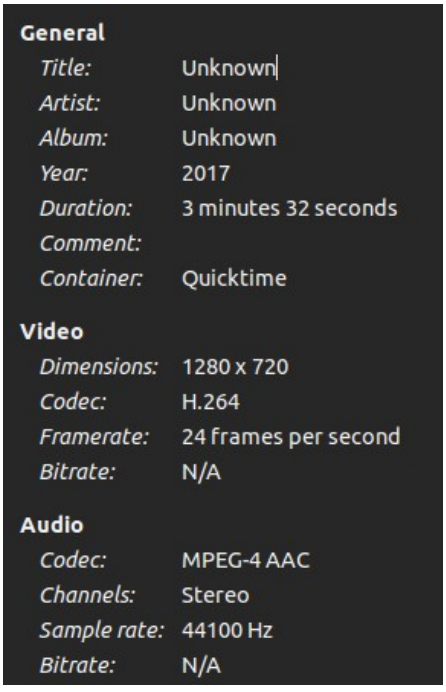


Imagem 2: Características do sample1.mp4

Vídeo: sample2.mp4
Tamanho: 175.3 MB (175,281,521 bytes)

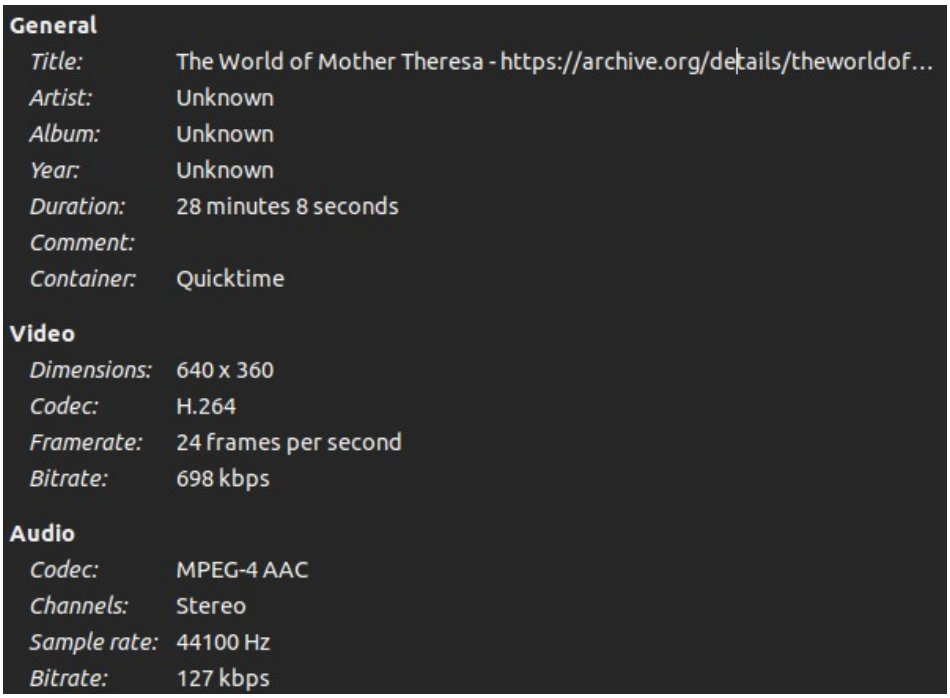


Imagem 3: Características do sample2.mp4

Video: sample3.mp4
Tamanho: 1.4 GB (1,438,663,096 bytes)

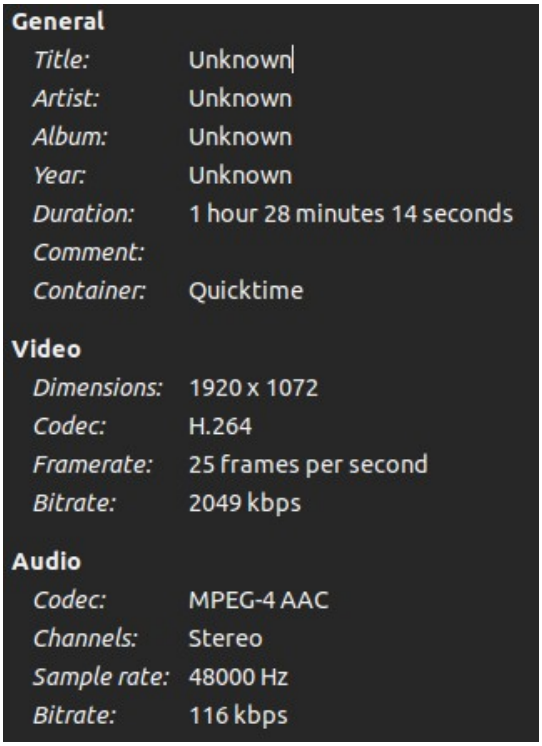


Imagem 4: Características do sample3.mp4

Foi tomado como primeiro exemplo de transmissão streaming o vídeo sample1.mp4:

Através do RTMPDump foi gerado automaticamente os dados de configuração do streaming, pela característica do vídeo que estava sendo enviado:

```
RTMPDump v2.4
(c) 2010 Andrej Stepanchuk, Howard Chu, The Flvstreamer Team; license: GPL
Connecting ...
INFO: Connected...
Starting download at: 0.000 kB
INFO: Metadata:
INFO:    width                  1280.00
INFO:    height                 720.00
INFO:    displayWidth           1280.00
INFO:    displayHeight          720.00
INFO:    duration               212.30
INFO:    videocodecid           7.00
INFO:    audiocodecid           10.00
INFO:    audiosamplerate        44100.00
31059.893 kB / 212.25 sec (99.9%)
Download complete
```

Para confirmar se o servidor estava realizando a transmissão de stream corretamente, foi feito um request através do VLC:

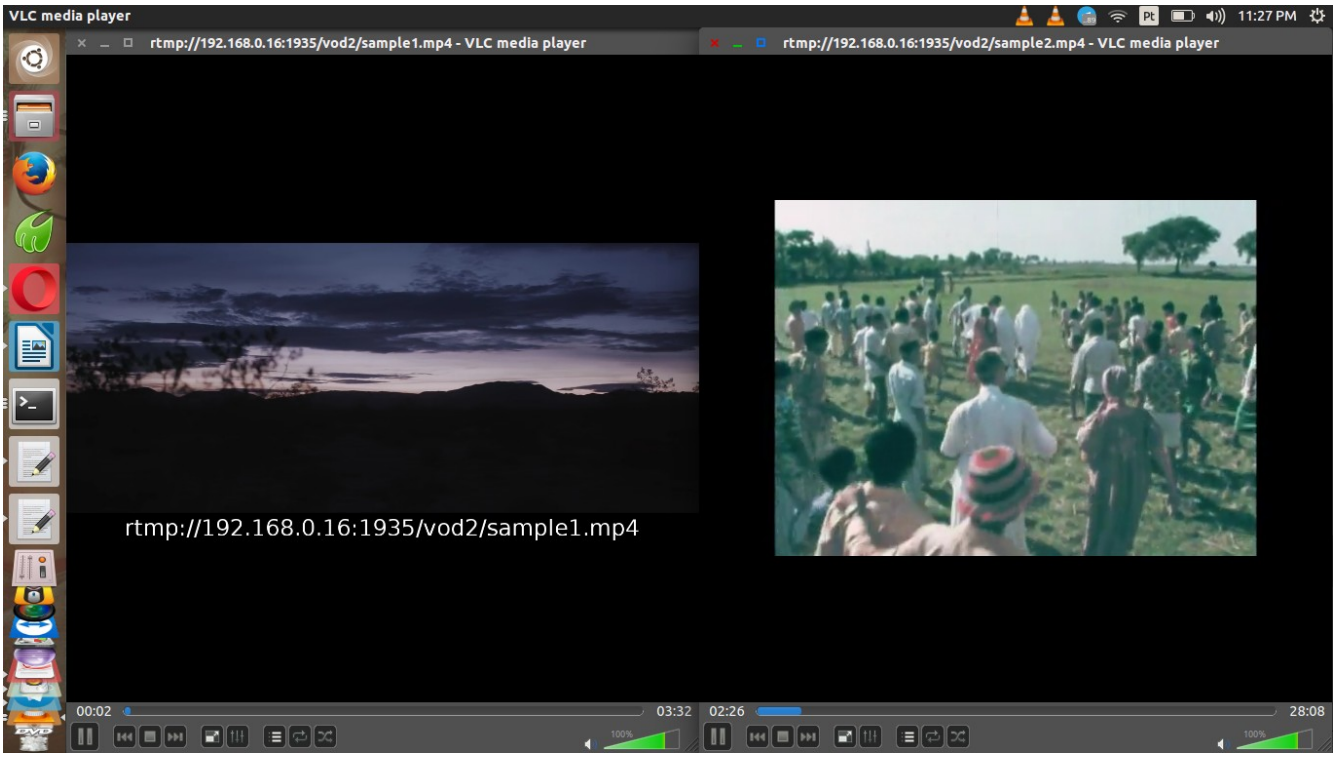


Imagem 5 – Stream de dois vídeos simultâneos

Em que, neste exemplo acima, foram utilizadas duas requisições de stream simultâneas, sem uma influenciar na outra. A verificação de duas transmissões foi somente com a finalidade de prova de

conceito.

Através da transmissão ao vivo do VLC, de um vídeo com as

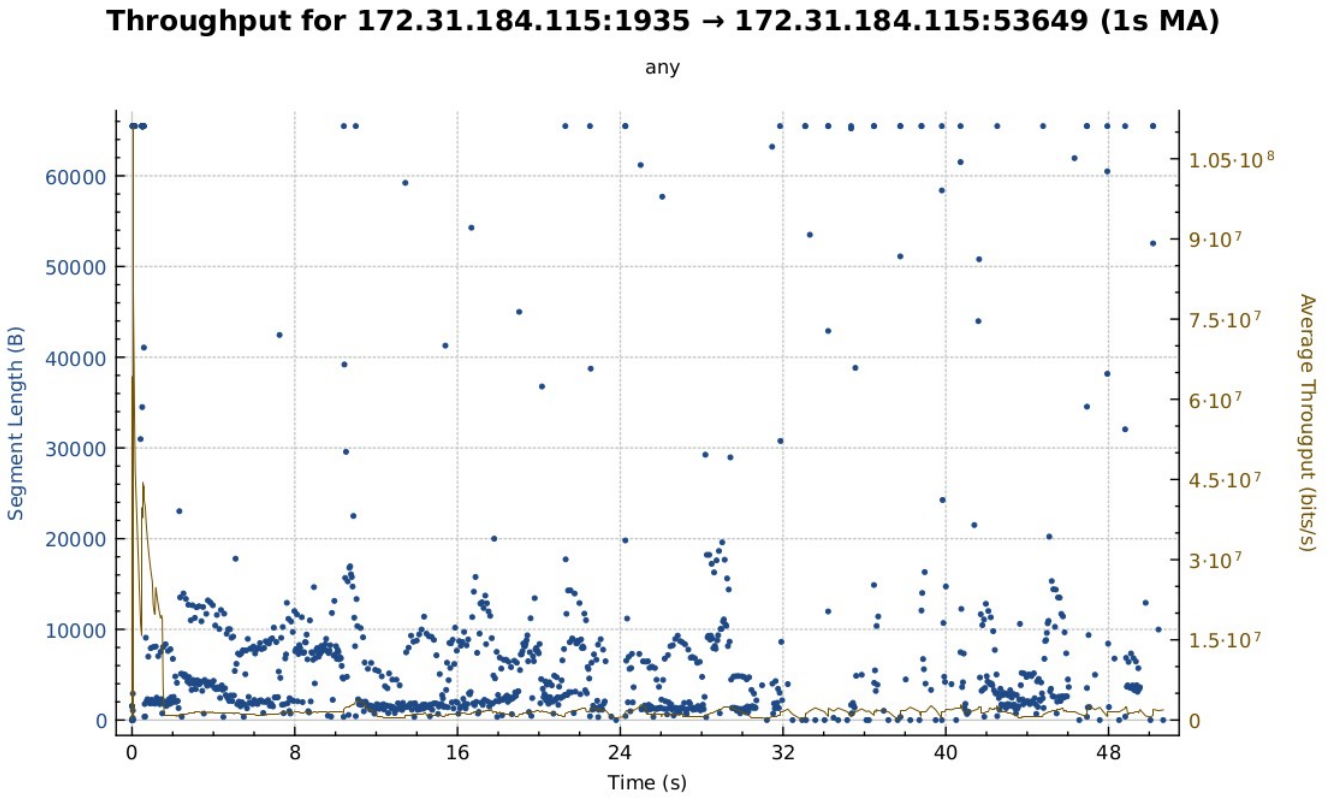


Imagem 6: Vazão do streaming do servidor para o cliente

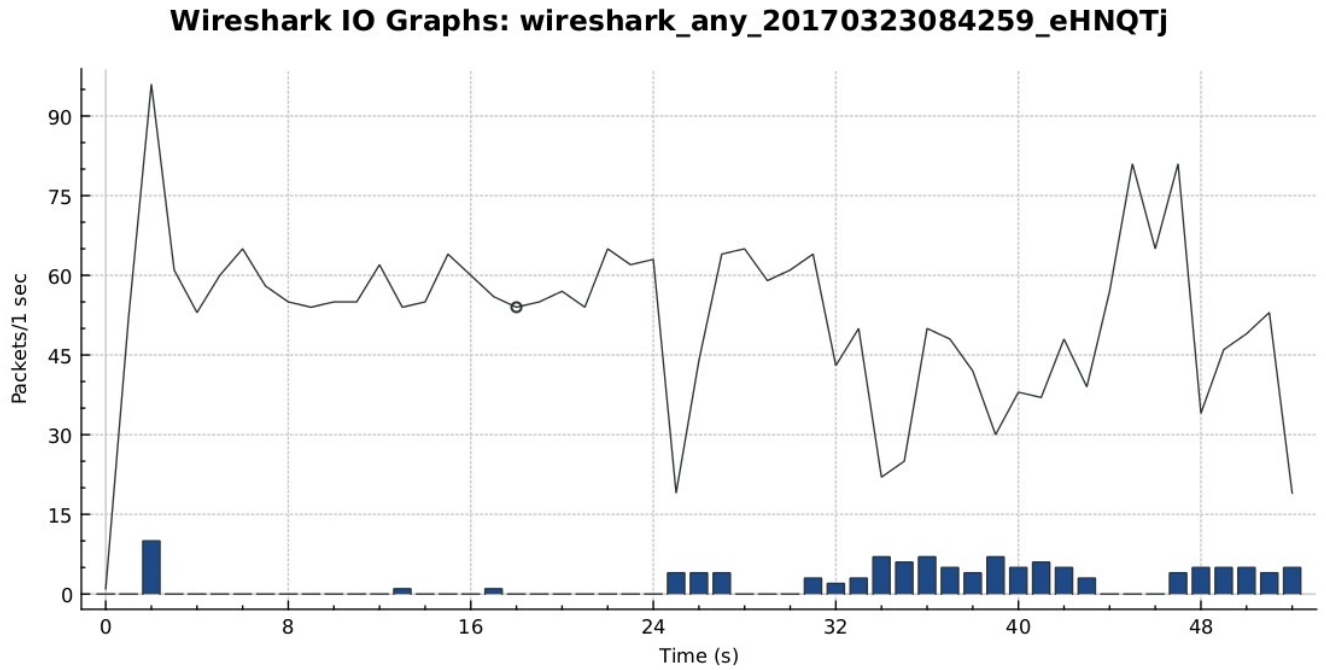


Imagem 7: Gráfico de I/O

Imagem 8: Flow Graph

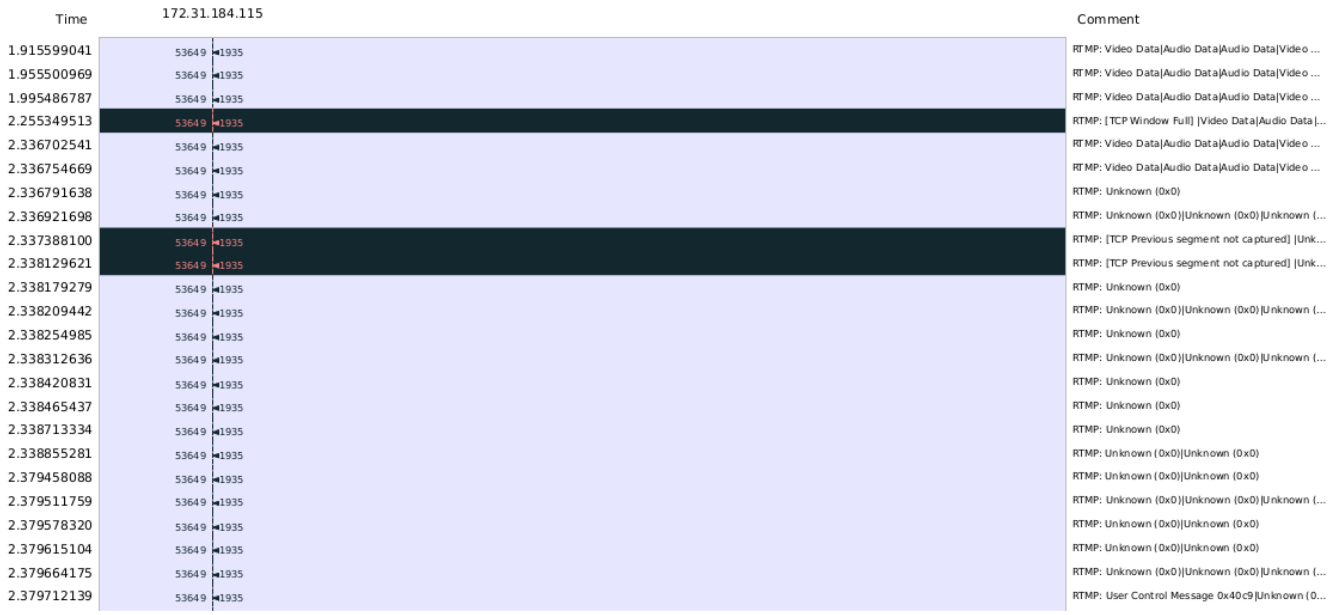


Imagem 9: Flow Graph

Na captura de pacotes foi possível identificar o Handshake descrito anteriormente, em que, no primeiro pacote enviado na transmissão, é apresentado os valores de C0+C1, como mostrado a seguir:

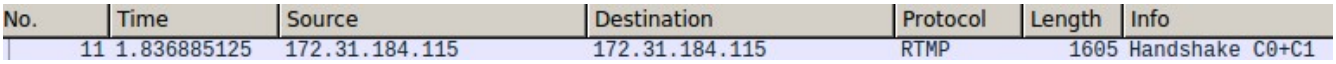


Imagem 10: Primeiro pacote RTMP - C0+C1

Imagem 11: Informações do pacote de número 11

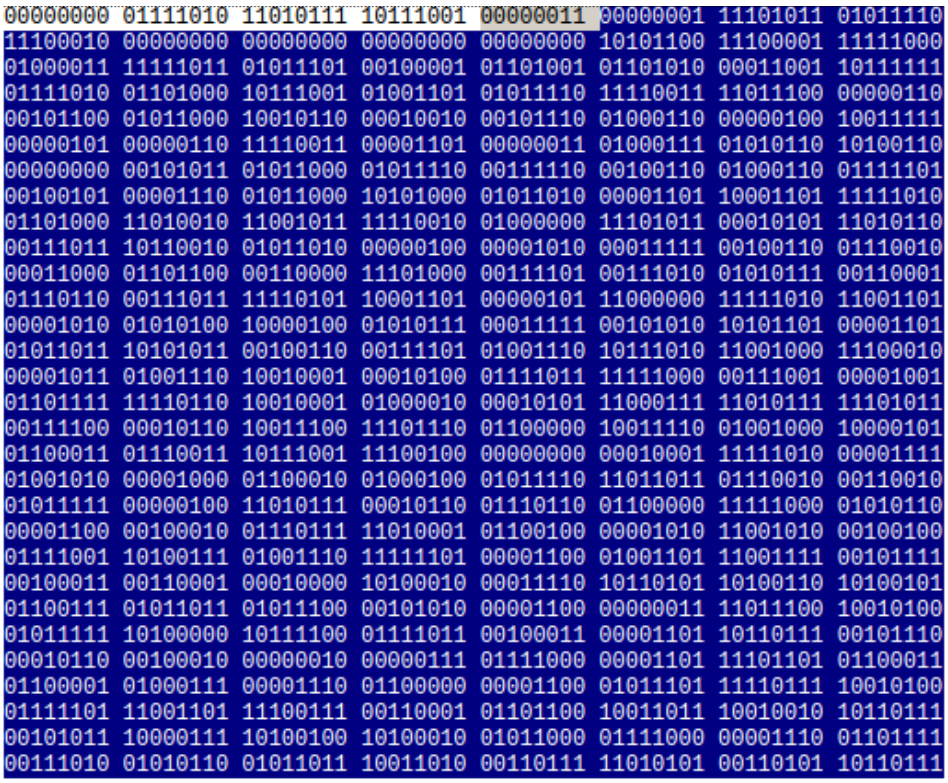


Imagem 12: Valores do pacote C1 marcados em azul

Em que, os primeiros 4 bytes são reservados ao tempo, sendo eles – 0x01EB5EE2 valores arbitrários. Os próximos 4 bytes são reservados a valores 0 e os próximos 1528 bytes são valores aleatórios.

Já no segundo pacote da transmissão, é enviada a resposta do servidor através do S0, S1 e S2, como descrito na imagem a seguir

No.	Time	Source	Destination	Protocol	Length	Info
15	1.837049147	172.31.184.115	172.31.184.115	RTMP	1604	Handshake S0+S1+S2

Imagem 13: Segundo pacote RTMP - S0+S1+S2

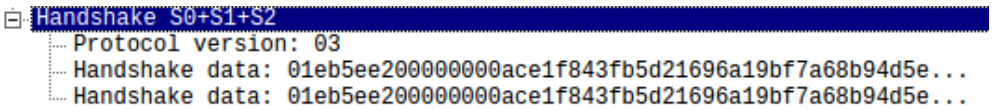


Imagem 14: Informações de pacote de número 15

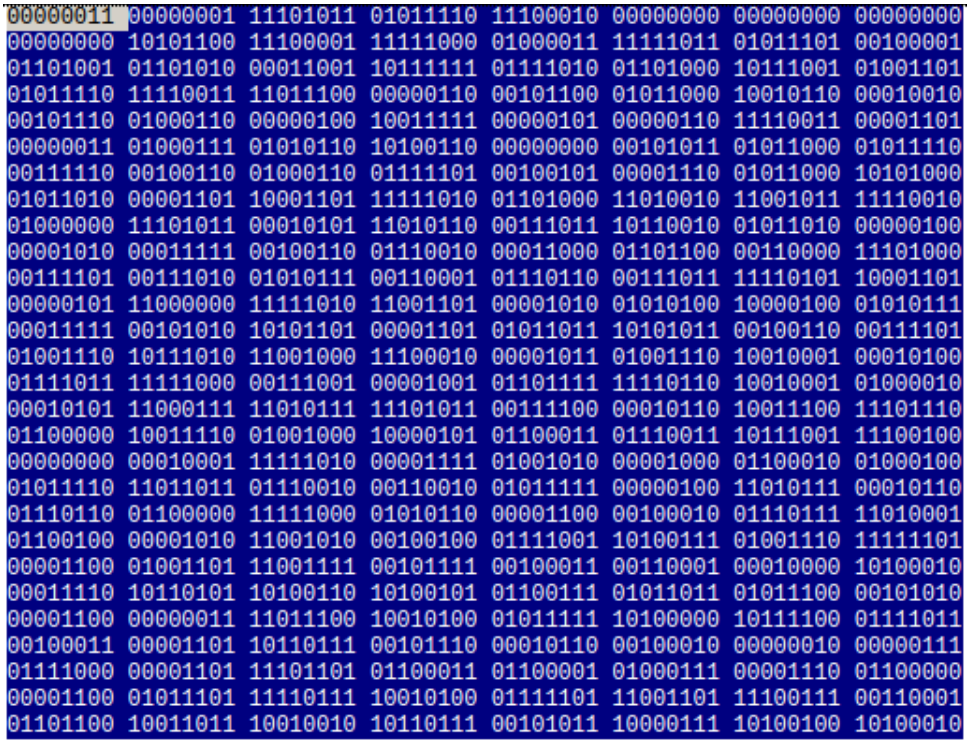


Imagem 15: Valores do pacote S1 marcados em azul

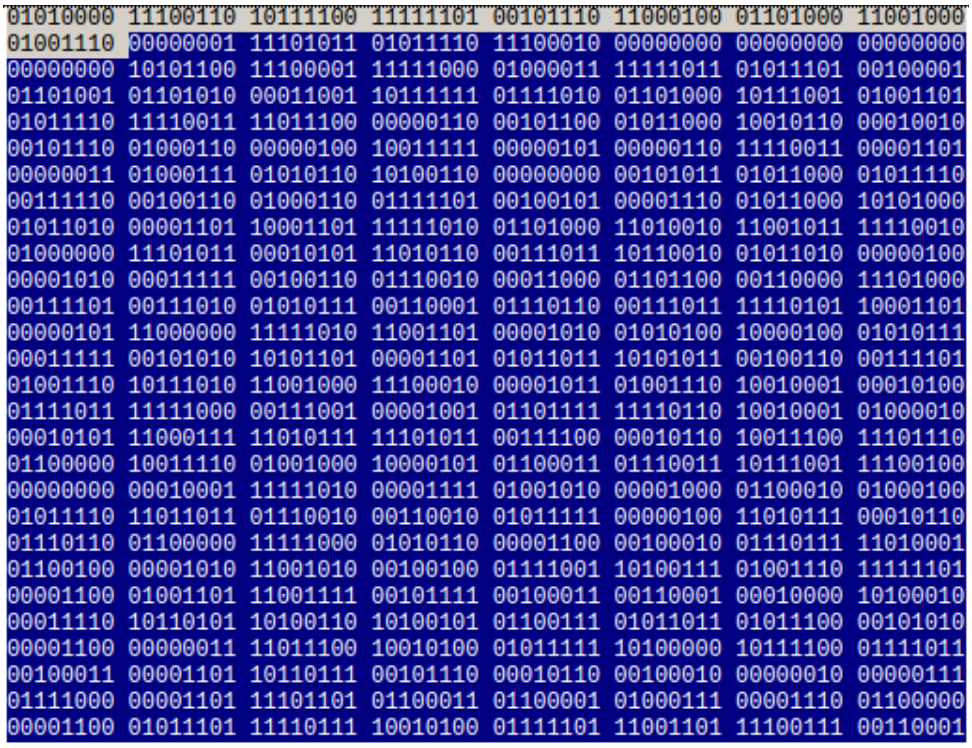


Imagem 16: Valores do pacote S2 marcados em azul

Em que, em S1, os primeiros 4 bytes são reservados ao tempo, sendo eles – 0x01EB5EE2, assim como o requisitado pelo C1. Os próximos 4 bytes são reservados a valores 0 e os próximos 1528 bytes são valores aleatórios.

Já em S2, os primeiros 4 bytes é o valor de tempo respondido pelo S1 e os próximos 4 bytes correspondem ao tempo e do envio do S1 ao cliente e os restantes 1528 bytes sendo valores aleatórios.

Referências

nginx: download

Configurações do servidor

Streaming Video on Demand with nginx and RTMP Module
Nginx-rtmp-module

Erros apresentados

connect() failed (111: Connection refused) while connecting to upstream

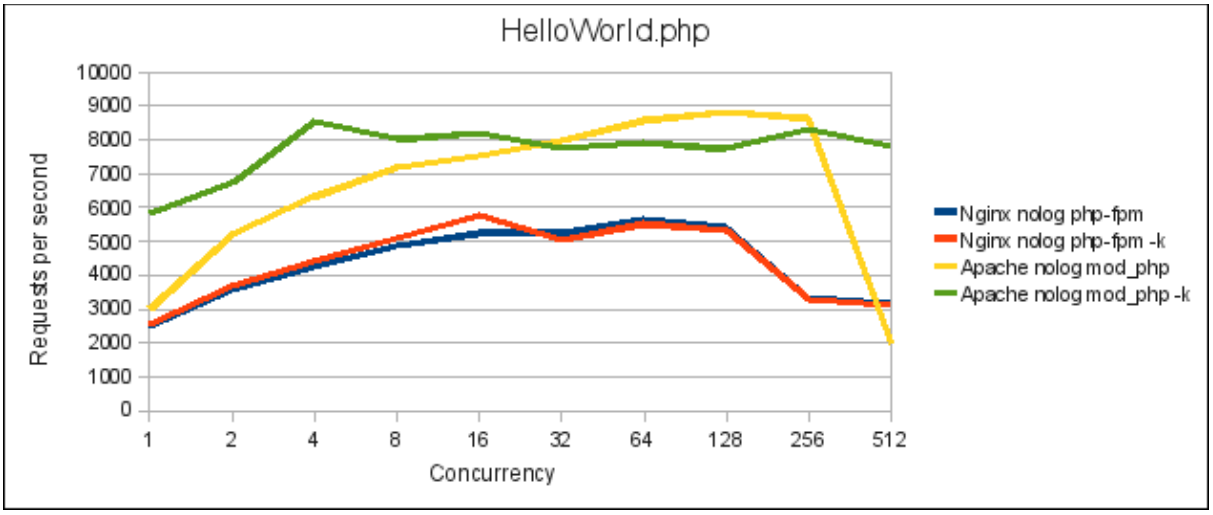
TO-DO
Procurar por broadcast stream nginx

Se der tempo, quando finalizar o servidor, verificar o php-fmp
Instalação

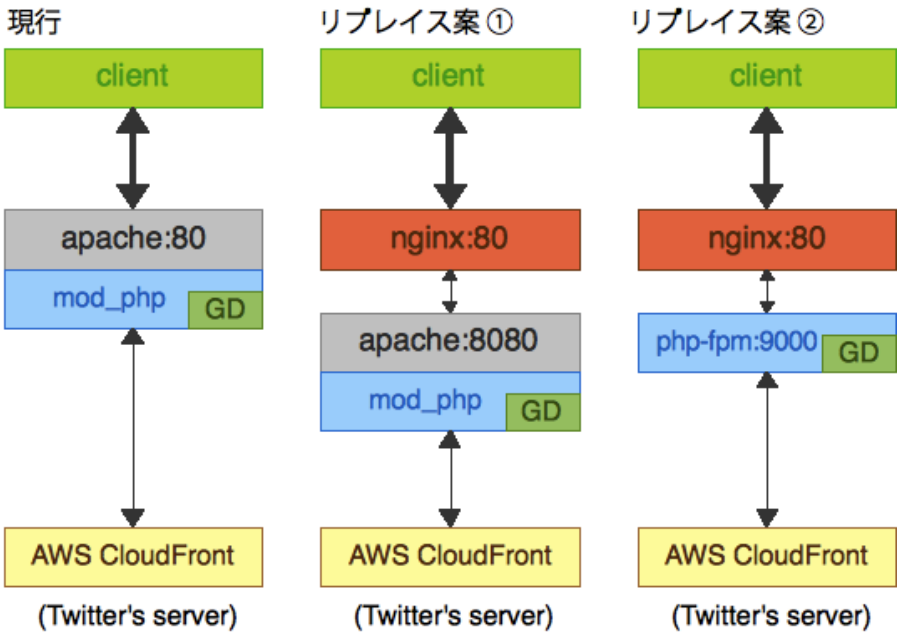
http://tutspundit.com/howto-install-nginx-php-fpm-mysql-php533-wordpress-ubuntu-part-1/

Demonstração de desempenho

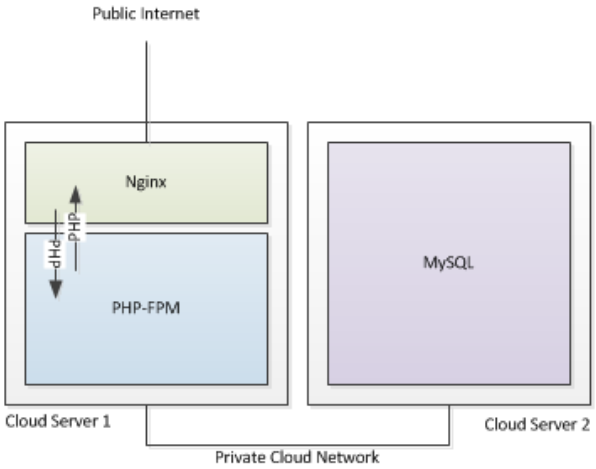
http://server-setting.info/centos/apache-nginx-2-rpm-php-fpm-install.html



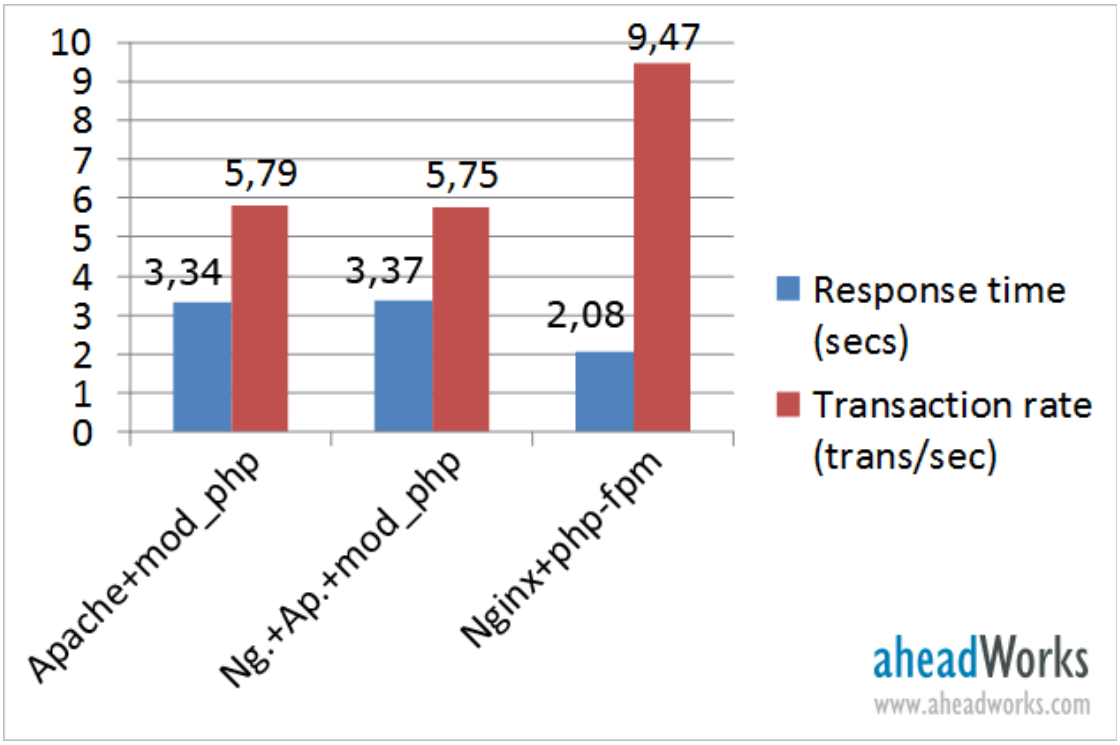
https://blog.a2o.si/2009/06/24/apache-mod_php-compared-to-nginx-php-fpm/



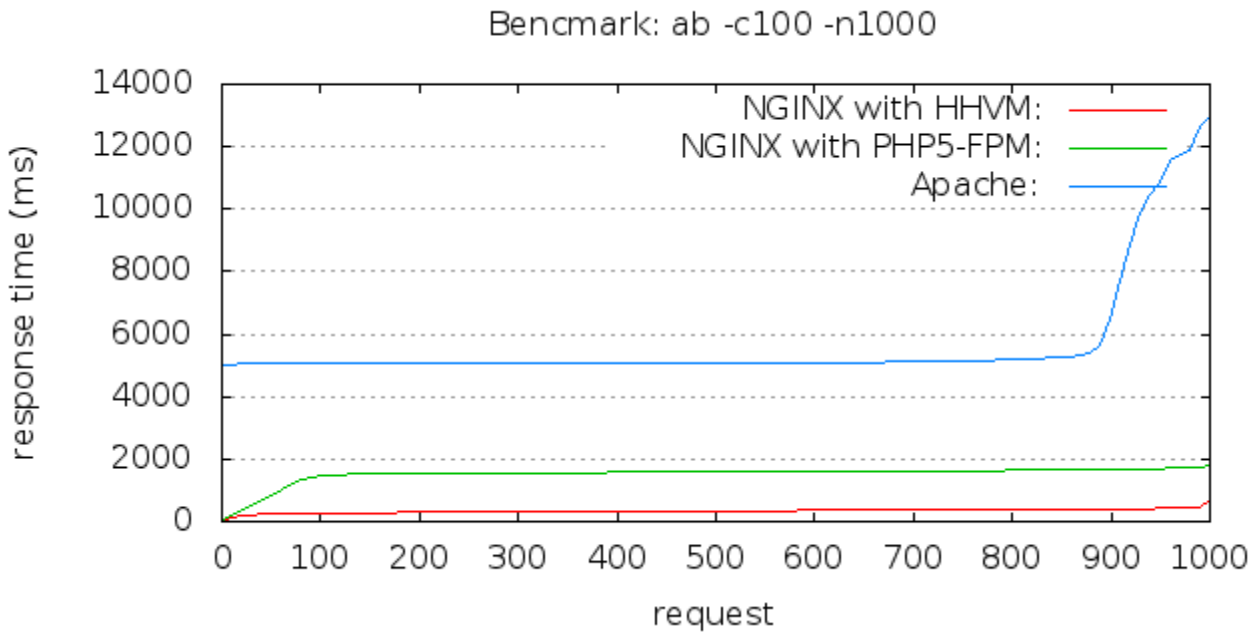
http://cloudrop.jp/tag/php-fpm



http://blog.corujadeti.com.br/performance-com-nginx-nao-e-so-cluster-e-seguranca/



http://magebase.com/magento-tutorials/optimizing-magento-performance/



https://makandracards.com/herdian-sc/30951-benchmark-between-nginx-with-hhvm-vs-nginx-with-php-fpm-vs-apache