UNIVERSIDADE FEDERAL FLUMINENSE

Instituto de ciência e tecnologia

Bacharelado em Ciência da Computação

Rafael Marques de Salles Soares e Savio Candido Martins

Protocolo RTMP: um projeto prático

Catalogação na fonte. UFF / SDC / Biblioteca de Rio das Ostras.

S676 2017

Soares , Rafael Marques de Salles. Protocolo RTMP: um projeto prático. / Rafael Marques de Salles Soares, Savio Candido Martins; Leandro Sousa, orientador. Rio das Ostras: s. n., 2017.

47 f.

Trabalho de conclusão de curso (Graduação em Ciência da Computação) – Universidade Federal Fluminense, Instituto de Ciência e Tecnologia, Departamento de Ciência da Computação. Campus de Rio das Ostras.

1. Protocolos de comunicação de redes. 2. Qualidade de serviço. 3.Produção intelectual. I. Título. II. Sousa, Leandro. (orientador).

CDD 22.ed. – 004.62

Rio das Ostras

2017

Rafael Marques de Salles Soares e Savio Candido Martins

Protocolo RTMP: um projeto prático

Trabalho de Conclusão de Curso submetido ao Curso de Ciência da Computação da Universidade Federal Fluminense como requisito parcial para obtenção do título de Barachel em Ciência da Computação.

Orientador:

Leandro Soares de Sousa

RIO DAS OSTRAS

2017

Rafael Marques de Salles Soares e Savio Candido Martins

Protocolo RTMP: um projeto prático.

Trabalho de Conclusão de Curso submetido ao Curso de Ciência da Computação da Universidade Federal Fluminense como requisito parcial para obtenção do título de Barachel em Ciência da Computação.

Rio das Ostras, 13 de Julho de 2017.

Banca Examinadora:

Leandro Soares de Sousa

Prof. Leandro Soares de Sousa, D.Sc. – Orientador

UFF - Universidade Federal Fluminense

Marcos Ribeiro Quinet de Andrade

Prof. Marcos Ribeiro Quinet de Andrade, D.Sc. – Avaliador

UFF - Universidade Federal Fluminense

André Renato V Silva

Prof. André Renato Villela da Silva, D.Sc. – Avaliador

UFF - Universidade Federal Fluminense

A nossa formação como profissional não poderia ter sido concretizada sem a ajuda de nossos queridos pais, que, nos proporcionaram, além de extenso carinho e amor, os conhecimentos da integridade, da perseverança. Por essa razão, gostaria de

dedicar e reconhecer a vocês, nossa imensa gratidão e sempre amor.

AGRADECIMENTOS

A todos os professores presentes em nossas vidas, que de certa forma participaram deste trabalho.

Aos professores Robson Britto e Fernando Salliby que apesar de lecionarem algumas das disciplinas mais árduas do nosso curso, nunca nos desmotivaram com as dificuldades.

Dalessandro Vianna que demonstrou através de suas aulas e exercicios o poder do esforço.

Aos professores Fábio Gonçalves e Fábio Ferreira de Freitas que tanto nos ensinaram nas disciplinas de matemática do curso.

Leandro Sousa que tanto nos auxiliou neste trabalho e em nenhum momento deixou as dificuldades nos abalar.

Agradeçemos a todos os amigos de curso, especialmente aos amigos, Diego Gomes, Erick Mendonça, Andrew de Castro, Matheus Adams, Roberto Nascimento, Flávio Telles, Sebastião Rodrigues Maia e Romulo Eduardo.

“Tudo é impossível para quem não sabe e possível para quem sabe."

Racional Superior - livro Universo em Desencanto

RESUMO

Tendo em vista o vasto aumento de serviços envolvendo conteúdos multimídia sob demanda, o objetivo desse trabalho é demonstrar vantagens e desvantagens do protocolo RTMP, se embasando em sua documentação.

Com o interesse de explorar seu desempenho e falhas foi desenvolvido um projeto que utiliza conceitos de Interface de programação de aplicativo e um servidor provedor de serviço de internet. Foram coletados todos os dados importantes em situações de cenários chave, realizando-se uma comparação desses resultados.

Através dos resultados obtidos nos experimentos, conclui-se que em um ambiente que disponha de uma melhor infraestrutura é perfeitamente viável a implantação desse projeto em um provedor de internet.

Levando em conta que *stream* de vídeo é um processo custoso ao servidor, o mais indicado seria manter uma estrutura mais elaborada para o próprio servidor RTMP dentro do servidor provedor de serviço de internet, ou seja, dois computadores funcionando em pararelo e não apenas um só, como foi exemplificado nesse projeto.

Palavras-chave: RTMP, API, Protocolo.

ABSTRACT

In view of the vast increase in services involving multimedia content on demand, the purpose of this work is to demonstrate the advantages and disadvantages of the RTMP protocol, based on its documentation.

With the interest of exploring its performance and failures, a project was built using API concepts and an ISP server. All important data were collected in specific situations, and a comparison of these results was made.

Through the results obtained in the experiments, it is concluded that in an environment that has a better infrastructure, it is perfectly feasible to implement this project in an internet provider with a reasonable number of clients.

Assuming that video stream is a costly process to the server, it would be more appropriate to have a more elaborate structure for the RTMP server inside the ISP, that is, two computers running in parallel, not just one, as exemplified in this project.

Keywords: RTMP, API, Protocol.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

ISP – *Internet Service Provider*

RTMP - *Real Time Messaging Protocol*

PPPoE - *Point-to-Point Protocol over Ethernet*

CHAP - *Challenge Handshake Authentication Protocol*

PPP - *Point-to-point Protocol*

TCP - *Transmission Control Protocol*

IP - *Internet Protocol*

API – *Application programming interface*

AMF - *Action Message Format*

SUMÁRIO

1 INTRODUÇÃO

O presente trabalho é um estudo de viabilidade de como integrar um servidor multimídia a um provedor de internet (ISP). Também apresenta possibilidades de visualização de conteúdos em tempo real, chamados *Live Stream*. Tudo se torna possível graças ao protocolo RTMP (ADOBE, 2016), a ser descrito nesse trabalho.

A principal motivação para o estudo e a implantação deste protocolo, foi a gama de possibilidades nas quais se pode aplicá-lo, tendo em vista que é possível utilizá-lo em vídeos sob demanda e em transmissões ao vivo, como é o caso dos principais *sites* de *Live Streams* de jogos do mundo, por exemplo, o Twitch e o Youtube Gaming.

Foram analisados os padrões do protocolo RTMP e com isso foi desenvolvida uma aplicação com um servidor NGINX (NGINX INC, 2017) a fim de demonstrar o funcionamento do servidor multimídia e do cliente. O NGINX é um servidor *web* rápido, leve e com inúmeras possibilidades de configuração para melhor performance. Esse servidor responde através de uma API que utilizada o padrão REST (TILKOV, 2008). Esta API foi escrita com a linguagem PHP (THE PHP GROUP, 2017) utilizando o *micro-framework* Lumen (LARAVEL, 2017). O Lumen é um *micro-framework* derivado do *framework* Laravel e seu principal objetivo é servir de base para aplicativos/serviços que dependam de performance para funcionar.

O conteúdo é exibido em um *site*, que utiliza a linguagem Javascript e o *framework* AngularJS, depois de tratado na API (GOOGLE, 2017).

O presente trabalho estrutura-se em seis capítulos. No segundo capítulo foram incluídas as principais definições e características acerca do protocolo RTMP, baseado em sua documentação, além de outras tecnologias onde o RTMP se faz dependente. O terceiro capítulo é dedicado a proposta de projeto utilizada para o desenvolvimento do trabalho, e também apresenta o ambiente e as tecnologias empregadas na construção do projeto. O quarto capítulo se foca em técnicas e

detalhes sobre a implementação de componentes do projeto. O quinto capítulo caracteriza o estudo de caso, com análises de desempenho em ocasiões distintas. No sexto capítulo são retratadas algumas limitações e conclusões provenientes das análises obtidas no capítulo cinco e também são relatadas algumas intenções e propostas de trabalhos futuros.

2

REVISÃO TEÓRICA

O *RTMP* é um protocolo que fornece um serviço de mensagensmultiplexadas e bidirecionais através de um fluxo de transporte confiável, destinado a transportar fluxos paralelos de vídeo, áudio e mensagens de dados (H.PARMAR e M.THORNBURGH,2012).

O *TCP* é utilizado como protocolo base para o *RTMP*, em casos gerais, para se realizar uma conexão através da internet. Poder-se-ia produzir este trabalho em uma rede local sem o uso de *PPPoE* e um *ISP*. Porém, esta delimitação iria gerar resultados bastantes divergentes da realidade. Baseando-se neste argumento, foi utilizado um *ISP* simulado, o protocolo *PPPoE* para conexão e alguma aplicação que implemente o *RTMP*.

A robustez do TCP e sua versatilidade o tornou apropriado à rede global, uma vez que este verifica se os dados são enviados de forma adequada, na sequência correta e sem erros.

O RTMP é está na camada de aplicação da pilha de protocolos, já o TCP é um protocolo de nível da camada de transporte da pilha de protocolos da internet e é sobre o qual que se assentam a maioria das aplicações da *World Wide Web*. O protocolo de controle de transmissão provê confiabilidade, entrega na sequência adequada e realiza verificação de erros para os pacotes de dados.

O PPPoE: (*Point-to-Point Protocol over Ethernet*) é um protocolo para conexão de usuários em uma rede *Ethernet* à Internet. O protocolo PPPoE deriva do protocolo PPP, estabelecendo a sessão e realiza a autenticação com o provedor de acesso a Internet. O PPP (*point-to-point protocol*) é um protocolo que permite o acesso autenticado e transmissão de pacotes de diversos protocolos, originalmente em conexões de ponto a ponto (como uma conexão serial). O PPP encapsula o protocolo TCP/IP, no acesso discado à internet. O protocolo PPPoE utiliza a tecnologia *Ethernet*, que é usada para ligar a placa de rede ao *modem*, ele faz a autenticação para a conexão e aquisição de um endereço IP fixo à máquina do usuário.

O protocolo PPPoE possui duas fases: a fase de descoberta e o estágio de sessão PPPoE. Na fase de descoberta, o cliente descobre o concentrador de acesso ao identificar o endereço MAC do mesmo e estabelecem uma sessão PPPoE, o concentrador de acesso PPPoE através do ISP concede acesso a um grande número de clientes sem precisar criar uma conexão dedicada para cada um. Na fase de estágio de sessão PPPoE, o cliente e o concentrador de acesso constroem uma conexão ponto-a-ponto sobre *Ethernet*, com base na informação recolhida na fase de descoberta.

O ISP, do inglês *Internet Service Provider*, é uma organização que oferece, principalmente, serviços de acesso à internet. No presente trabalho utilizou-se um ISP simulado.

2.1 DEFINIÇÕES DE PROTOCOLO

São apresentados a seguir algumas definições relevantes para o entendimento do protocolo:

- **Pacote**: um pacote de dados é constituído por dados de cabeçalho e o *Payload*. Alguns protocolos podem exigir um encapsulamento dos pacotes.
- ***Payload***: são dados contidos em um pacote, por exemplo, amostras de áudio ou dados de vídeo comprimido. É o conteúdo a ser transmitido, que é o principal objetivo da transmissão, excluindo os dados complementares (como cabeçalhos ou metadados, que podem conter, dentre outras informações, a identificação da fonte e do destino dos dados) apenas para facilitar a entrega.
- **Porta**: é uma abstração que os protocolos usam para diferenciar qual destino um pacote terá dentro de um *host*.

- ***Transport Address***: a combinação de um endereço de rede e porta que identifica um ponto de extremidade no nível de transporte, no caso da internet esse endereço é representado por um endereço IP e uma porta TCP.
- ***Chunk***: um fragmento de uma mensagem. As mensagens são divididas em partes menores e intercalados antes de serem enviados através da rede.
- ***Multiplexing***: processo de tornar os dados de áudio / vídeo separados em um fluxo de áudio / vídeo coerente, tornando-se possível transmitir vários vídeos e áudios simultaneamente.

2.2 CARACTERÍSTICAS DO RTMP

Uma mensagem que pode ser dividida em *Chunks,* para suportar multiplexação, depende de um protocolo de nível mais elevado. A mensagem deve, contudo, conter os seguintes campos que são necessários para criar os *Chunks*.

Timestamp: este campo pode transportar 4 *bytes* e indica o momento em que a mensagem foi criada.
Length: tamanho de cada *Payload*, se o cabeçalho da mensagem não puder ser excluído, ele deve ser incluído neste campo. Este campo ocupa 3 *bytes* no cabeçalho e indica a carga útil da mensagem.
Type ID: campo que define qual o tipo de dado será passado naquele pacote, dentre todas as seguintes possibilidades
0x01 = Pacote que estipula o tamanho da mensagem.
0x04 = Mensagem de *Ping*.
0x05 = A largura de banda do servidor.
0x06 = A largura de banda do cliente.
0x08 = Pacote de áudio.

0x09 = Pacote de vídeo.

0x11 = Um comando do tipo AMF3.

0x14 = Um comando do tipo AMF0.

Message Stream ID: *O Message Stream ID* pode ser qualquer valor arbitrário. Diferentes *streams* são multiplexadas para o mesmo *chunk stream* e são demultiplexedas com base em seus *Message Stream ID* (H.PARMAR e M.THORNBURGH, p.6, 2012).

2.2.1 RTMP *Chunk Stream*

Fornece serviços de multiplexação e empacotamento para um protocolo de fluxo multimídia de alto nível. O *RTMP Chunk Stream* foi projetado para trabalhar com o *RTMP.* No entanto, ele pode lidar com qualquer protocolo que envia um fluxo de mensagens. *RTMP Chunk Stream* e o *RTMP* juntos são adequados para uma ampla variedade de aplicações de áudio e vídeo, a partir de uma transmissão ao vivo ou vídeo *on demand*.

Quando usado com um protocolo de transporte confiável, como o *TCP*, *RTMP Chunk Stream* garante a entrega de todas as mensagens, em vários *streams*. O *RTMP Chunk Stream* não fornece qualquer priorização ou formas similares de controle, mas pode ser usado por protocolos de nível superior para fornecer essa prioridade. Por exemplo, um servidor de vídeo ao vivo pode optar por deixar de enviar mensagens de vídeo para um cliente lento para garantir que as mensagens de áudio sejam recebidas em tempo hábil, baseada ou na hora de enviar ou no tempo para reconhecer cada mensagem. (H.PARMAR e M.THORNBURGH,2012).

2.2.2 Diagrama de Handshake

A Figura 1 apresenta através de um diagrama de sequência UML, como funciona a troca de pacotes no *handshake* do protocolo RTMP.

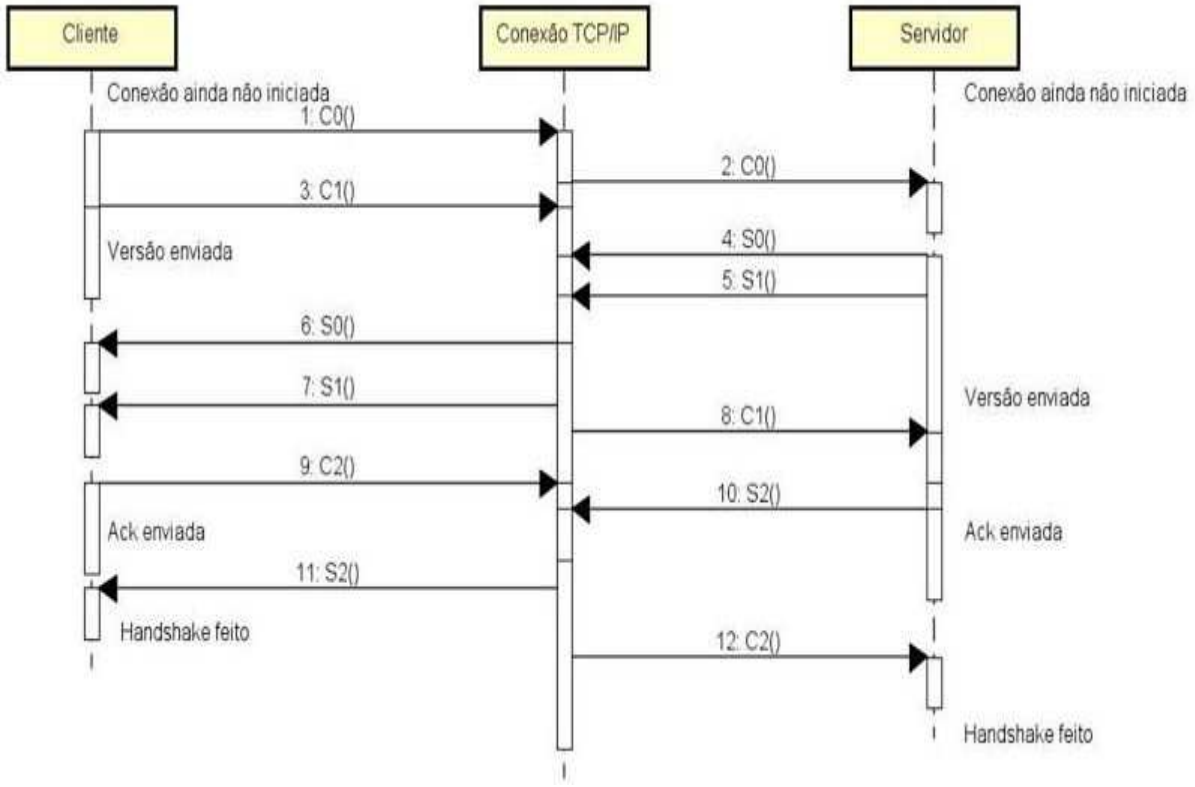

Figura 1: Diagrama de *Handshake*.

1.1.1 *Handshake*

Uma conexão *RTMP* começa com um *Handshake*. O *Handshake* é diferente do resto do protocolo, que consiste em três pares de *Chunks* de tamanho estáticos. O cliente e o servidor enviam cada um os mesmos três *Chunks*. Para exemplificar, esses *Chunks* serão denominados C0, C1 e C2, os *Chunks* enviados pelo cliente e S0, S1 e S2 os *chunks* enviados pelo lado do servidor (H.PARMAR e M.THORNBURGH,2012).

1.1.1.1 Sequência do *Handshake*

O *Handshake* começa com o cliente enviando os *Chunks* C0 e C1. O cliente deve esperar o *chunk* S1 ser recebido para enviar C2. O cliente deve esperar o *chunk* S2 ser recebido para enviar quaisquer outros dados.

O servidor deve esperar que o *chunk* C0 seja recebido para enviar S0 e S1, podendo também esperar o *chunk* C1 para o envio de S0 e S1. O servidor deve esperar até que C1 seja recebido para enviar S2. O servidor deve esperar até que o C2 seja recebido para enviar quaisquer outros dados.

1.1.1.2 Formatos do S0 e C0

Esses chunks são tratados como um inteiro de 8 *bits*. Possuem apenas 1 campo que é a versão, no *chunk* C0 este campo identifica a versão *RTMP* solicitada pelo cliente. No S0, este campo identifica a versão *RTMP* selecionada pelo servidor.

Um servidor que não reconhece a versão solicitada do cliente deve responder com a sua versão atual. O cliente pode escolher adequar-se à versão recebida pelo servidor ou abandonar o *Handshake*.

1.1.1.3 Formatos do S1 e C1

Os pacotes C1 e S1 são constituídos pelos seguintes campos:

***Time* (4 *bytes*)**: Este campo contém um *timestamp*, que deve ser usado como o momento atual para que todos os futuros *chunks* sejam enviados a partir deste *timestamp,* em algumas aplicações de terceiros este campo é definido como 0.
***Zero* (4 *bytes*)**: Esse campo deve ser todo 0.
***Random data* (1528 *bytes*)**: Este campo pode conter quaisquer valores aleatórios, serve principalmente como uma forma dos *endpoints* diferenciar a resposta do *handshake* inicial e o *handshake* iniciado por seu *peer*.

1.1.1.4 Formatos do S2 e C2

Os pacotes S2 e C2 são muito parecidos com os pacotes C1 e S1 respectivamente, pode-se dizer que S2 e C2 são quase um eco de C1 e S1, formados pelos seguintes campos:

Time (4 bytes): Este campo contém o *timestamp* enviado pelo *peer* de S1 para C2 ou de C1 para S2.
Time 2 (4 bytes): Este campo contém o *timestamp* contido no último pacote anterior (S1 ou C1) lido pelo *peer*.
Random echo (1528 bytes): Este campo contém dados aleatórios, normalmente é enviado pelo *peer* em S1 para C2 ou em S2 para C1, pode se usar os campos de

time e *time 2* juntos ao *timestamp* atual com o objetivo de se obter uma estimativa rápida de largura de banda ou latência de conexão, mas isso é improvável que seja útil.

1.1.2 *CHUNKING*

Após *o handshaking*, a conexão transmite simultaneamente uma *Chunk stream* ou mais. Cada *Chunk* stream transporta mensagens de tipo específico vindo de um *message stream*. O *Chunk* que é criado tem um *ID* exclusivo associado a ele, este é chamado de *chunk stream ID*. Durante a transmissão, cada *chunk* deve ser enviado completamente antes do envio do próximo. No final da recepção, os *chunks* são montados em mensagens baseadas em seus *chunk stream ID*.

O *Chunking* permite que grandes mensagens de algum protocolo de nível superior sejam divididas em mensagens menores, para evitar que grandes mensagens de vídeo bloqueiem mensagens de áudio por exemplo.

O *Chunking* também permite que mensagens pequenas sejam enviadas com menos sobrecarga pois o cabeçalho do *chunk* contém uma representação compactada de informações. Sem esta compactação estes dados teriam de ser incluídos na própria mensagem.

O tamanho do *chunk* é configurável, podendo ser definido usando o s*et chunk size control message*. Grandes *chunks* reduzem o uso de processamento, porém, por possuir arquivos maiores, podem atrasar outros conteúdos em conexões mais lentas. *Chunks* pequenos não são bons para *streaming* de alto fluxo de *bits.* (H.PARMAR e M.THORNBURGH,2012).

1.1.2.1 Formato dos *Chunks*

Cada *chunk* consiste em um cabeçalho e dados. O cabeçalho tem três partes:

- **Cabeçalho Básico (1 a 3 *bytes*)**: Este campo codifica o *chunk stream ID* e o tipo de *chunk*. O tipo de *chunk* determina o formato do cabeçalho da mensagem codificada. O tamanho do cabeçalho básico depende inteiramente do *chunk stream ID* que por sua vez é um campo de comprimento variável.
- **Cabeçalho da Mensagem (0, 3, 7 ou 11 *bytes*)**: Este campo codifica informações sobre a mensagem a ser enviada (no todo ou em parte). Seu comprimento pode ser determinado usando o tipo de *chunk* especificado no cabeçalho de *chunk*.
- ***Extended Timestamp* (0 ou 4 bytes)**: Este campo está presente apenas em certas circunstâncias, dependendo do *timestamp* utilizado ou do campo de timestamp que está no cabeçalho *chunk*.
- **Dados de *chunk* (tamanho variável)**: O tamanho destes dados pode alcançar no máximo o tamanho do maior *chunk* configurado.

1.1.2.2 Cabeçalho Básico do *Chunk*

O Cabeçalho básico do *chunk* codifica o *chunk stream ID* e o tipo de *chunk*. O tipo de *chunk* determina o formato do cabeçalho da mensagem codificada, o campo do cabeçalho básico do *chunk* pode conter 1, 2, ou 3 *bytes*, dependendo do *chunk stream ID*.

O protocolo suporta até 65597 *streams* com *IDs* entre 3 e 65599. Os *IDs* 0, 1, e 2 estão reservados.

O valor 0 indica a forma de 2 *bytes* e um *ID* no intervalo de 64-319 (o segundo *byte* + 64). Valor 1 indica a forma de 3 *bytes* e um *ID* no intervalo de 64 a 65599 ((o terceiro *byte*) * 256 + o segundo *byte* + 64). Os valores no intervalo de 3 a 63 representam o *stream ID* completo.

O *chunk stream ID* com valor 2 é reservado para mensagens e comandos de controle de protocolos em nível inferior, os *bits* 0-5 (menos significativos) no cabeçalho básico do *chunk* representam a identificação do *chunk stream ID*, existem 3 cabeçalhos básicos de *chunk*.

1.1.2.3 Cabeçalho de Mensagem Chunk

Existem quatro formatos diferentes para o cabeçalho de mensagem *Chunk*, selecionados através do campo "*fmt*" no cabeçalho básico do *chunk*. Uma implementação deve obrigatoriamente usar a representação mais compacta possível para cada cabeçalho de mensagem *Chunk*.

1.1.2.4 *Extended Timestamp*

O campo de *Extended Timestamp* é usado para codificar *timestamps* ou *delta timestamps* maiores que 16777215 (0xFFFFFF), isto é, para *timestamps* ou *delta timestamps* que não se encaixam no campo de 24 *bits* dos *Chunks* tipo-0, tipo-1, ou ambos. Este campo codifica o *timestamp* ou *delta timestamp* de 32 *bits*. A presença deste campo é indicada pela configuração de campo de *timestamp* no *Chunk* de tipo-0 ou para o campo de *delta timestamp* nos *chunks* de tipo 1 e 2. Este campo está presente no *chunk* de tipo 3 quando o *chunk* mais recente de tipo 0,1 ou

os 2 para o mesmo *Chunk Stream ID* indica a presença de um campo de *extended timestamp.* (H.PARMAR e M.THORNBURGH,2012).

1.1.3 Mensagens de Controle de Protocolo

De acordo com as definições de H.PARMAR e M.THORNBURGH:

> *"RTMP Chunk Stream* usa *Ids* de mensagens do tipo 1, 2, 3, 5 e 6 para o protocolo de controle de mensagens. Essas mensagens contêm informações necessárias para o protocolo *RTMP Chunk Stream*.
>
> Essas mensagens de controle de protocolo devem ter *Stream ID* de mensagens 0 (conhecido Como o *Stream* de controle) e deve ser enviado em um *Chunk Stream ID* 2. As mensagens de controle de protocolo entram em vigor assim que são recebidas, Seu *timestamp* é ignorado." (H.PARMAR e M.THORNBURGH,2012).

1.1.4 Formato da mensagem RTMP

O servidor e o cliente enviam mensagens *RTMP* através da rede para comunicar-se uns com os outros. As mensagens podem incluir áudio, vídeo, dados ou quaisquer outras mensagens. A mensagem *RTMP* tem duas partes, um cabeçalho (*Header)* e sua carga útil (*Payload*). (H.PARMAR e M.THORNBURGH,2012).

1.1.4.1 Cabeçalho da mensagem (*Header)*

O cabeçalho da mensagem contém o seguinte:

- **Tipo de Mensagem**: Um campo de *byte* para representar o tipo de mensagem. Um intervalo de *type IDs* entre 1 e 6 é reservado para mensagens de controle de protocolo.
- **Comprimento**: Campo de três *bytes* que representa o tamanho da carga (*Payload*) em *bytes* é definido no formato *big-endian*.
- ***Timestamp***: Campo de quatro *bytes* que contém um *Timestamp* da mensagem. Os 4 *bytes* são empacotados no formato *big-endian*.
- ***Message Stream Id***: Campo de três *bytes* que identifica o *Stream* da mensagem. Estes *bytes* são definidos no formato *big-endian*.

1.1.4.2 Carga útil da Mensagem (*Message Payload*)

A outra parte da mensagem é a carga útil, são os dados contidos na mensagem. Por exemplo, pode ser amostras de áudio ou dados de vídeo compactados. O formato da carga e a sua interpretação estão fora do escopo deste documento.

1.1.5 Mensagens de Controle de Usuário

O *RTMP* usa o *Type ID* de mensagem 4 para mensagens de Controle do Usuário. Essas mensagens contêm informações usadas pela camada *RTMP streaming*. As mensagens de protocolo com *IDs* 1, 2, 3, 5 e 6 são usadas pelo protocolo *RTMP Chunk Stream*.

O cliente ou o servidor envia esta mensagem para notificar o *peer* sobre os eventos de controle do usuário. Esta mensagem transporta dois campos:

Os primeiros 2 *bytes* dos dados da mensagem são usados para identificar o tipo de evento (*Event Type*). O tipo de evento é seguido por dados do evento

(*Event Data*). O tamanho do campo ‘Dados’ do evento é variável. No entanto, nos casos em que a mensagem tem de passar através da camada RTMP *Chunk Stream*, o tamanho máximo do pedaço deve ser grande o suficiente para permitir que essas mensagens se encaixem em um único pedaço (H.PARMAR e M.THORNBURGH,2012).

1.1.6 Mensagens de Comando RTMP

Esta seção descreve os diferentes tipos de mensagens e comandos que são trocados entre o servidor e o cliente para se comunicar. Os diferentes tipos de mensagens que são trocadas entre o servidor e o cliente incluem mensagens de áudio, mensagens de vídeo, mensagens de dados referente a algum dado enviado pelo usuário, mensagens de objeto compartilhado e mensagens de comando. Mensagens de objeto compartilhado fornecem uma maneira geral de gerenciar os dados entre vários clientes e um servidor. As mensagens de comando transportam comandos AMF codificados entre o cliente e o servidor. Um cliente ou um servidor pode solicitar chamadas de procedimento remoto (RPC) sobre fluxos que são comunicadas usando as mensagens de comando para o *peer*. (H.PARMAR e M.THORNBURGH,2012).

1.1.6.1 Mensagem de Comando (tipo 20 e 17)

As mensagens de comando carregam os comandos AMF codificados entre o cliente e o servidor. A estas mensagens foi atribuído o valor do tipo de mensagem 20 para codificação AMF0 e 17 para AMF3. Estas mensagens são enviadas para executar algumas operações como *Connect*, *Create Stream*, publicar, reproduzir, pausar no *peer*. Mensagens de Comando como *onstatus*, resultado, etc,

são usadas para informar o remetente sobre o *status* dos comandos solicitados. Uma mensagem de comando consiste no nome do comando, *ID* da transação e objeto de comando que contém parâmetros relacionados. Um cliente ou um servidor pode solicitar chamadas de procedimento remoto (*RPC*) sobre os fluxos que são comunicados usando as mensagens de comando para o *peer*.

1.1.6.2 Mensagem de Dados (tipo 18 e 15)

O cliente ou o servidor envia esta mensagem para enviar metadados ou qualquer dado do usuário para o *peer*. Os metadados incluem detalhes sobre os dados (áudio, vídeo, etc.) como tempo de criação, duração, tema e assim por diante. A estas mensagens foi atribuído um valor de tipo de mensagem de 18 para AMF0 e 15 para AMF3.

1.1.6.3 Mensagem de Objeto Compartilhado (tipo 19 e 16)

Um objeto compartilhado é um objeto *Flash* que estão em sincronização entre vários clientes e instâncias. Os tipos de mensagem 19 para AMF0 e 16 para AMF3 são reservados para eventos de objetos compartilhados. Cada mensagem pode conter vários eventos.

Os seguintes tipos de evento são suportados:

Tabela 1: Tabela de eventos.

Evento	Descrição
Utilização (= 1)	O cliente envia este evento para informar ao servidor sobre a criação de um objeto compartilhado nomeado.

Liberação (= 2)	O cliente envia este evento para o servidor quando o objeto compartilhado é eliminado no lado do cliente.
Alteração de solicitação (=3)	O cliente envia esse evento para solicitar a alteração do valor associado a um parâmetro nomeado do objeto compartilhado.
Alteração (= 4)	O servidor envia este evento para notificar todos os clientes, exceto o cliente que originou a solicitação. A notificação é sobre uma alteração no valor de um parâmetro nomeado.
Sucesso (= 5)	O servidor envia este evento para o cliente solicitante em resposta ao evento de alteração de solicitação se a solicitação for aceita.
Enviar mensagem (=6)	O cliente envia este evento para o servidor para transmitir uma mensagem. Ao receber este evento, o servidor transmite uma mensagem para todos os clientes, incluindo o remetente.
Status (=7)	O servidor envia este evento para notificar os clientes sobre as condições de erro.
Limpar (=8)	O servidor envia esse evento para o cliente para limpar um objeto compartilhado. O servidor também envia este evento em resposta ao evento usado no envio feito pelo cliente ao se conectar.

2

PROPOSTA

Após análise do RTMP, propôs-se nesse projeto simular um ISP e fazer com que os clientes se conectem por PPPOE para acessem um conteúdo multimídia armazenado no próprio ISP.

A motivação deste projeto foi o notório crescimento de plataformas de vídeos *on demand* como o NetFlix. A principal diferença está na velocidade em que os vídeos seriam carregados e na simplicidade de acessá-los, já que com essa arquitetura, os clientes não precisam ter acesso à Internet, somente uma conexão autenticada com o ISP. Neste capítulo serão mostradas as ferramentas e como elas funcionam.

2.1 AMBIENTE

O servidor foi criado utilizando um *notebook* Samsung com processador Intel Core i3, com 4gb de memória RAM e um sistema operacional Ubuntu 14.04 LTS de 64 *bits*, também foi implantado um servidor Web e também RTMP, NGINX.

A escolha do Ubuntu se deu pela facilidade de se achar conteúdo de programação dessa área. Com isso foi implantado o protocolo PPPOE com autenticação CHAP, que é um protocolo de autenticação, no próprio sistema operacional. O objetivo dessa implantação era fazer com que o *notebook* se conectasse aos outros clientes através da rede cabeada e concedesse recursos da rede através da autenticação CHAP. Com relação a Internet, foi utilizado um encaminhamento de pacotes da rede Wifi para a rede Ethernet. Com isso pronto, foi possível simular um pequeno ISP.

Com o ISP funcionando, fez-se a escolha do NGINX por ser um servidor web e RTMP rápido, leve e principalmente por sua compatibilidade com o sistema operacional escolhido. Sua configuração e simples permite com que sejam

configuradas muitas funcionalidades. Por padrão o NGINX exibe uma página HTML no *localhost*, essa página HTML que será acessada pelos clientes conectados via PPPOE, então fez-se necessário a implementação de uma página Web com o conteúdo a ser exibido.

Para essa página HTML, utilizamos o poderoso *framework* AngularJS e o media player JW Player, pela sua facilidade de utilização e personalização.

A Figura 2 mostra o funcionamento da aplicação deste projeto.

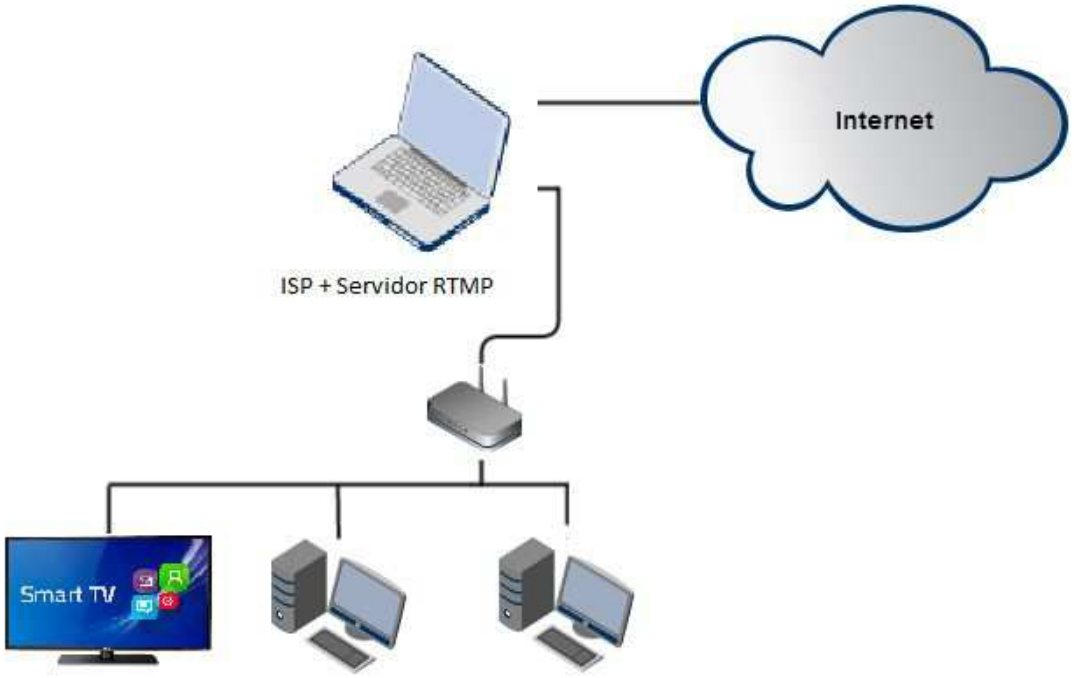

Figura 2: Arquitetura projetada para esse trabalho.

2.2 NGINX

NGINX é um servidor *Web* e RTMP, com muitas possibilidades de manipulação para atingir um maior desempenho de acordo com o que se necessita.

Na prática, o NGINX é amplamente utilizado pelo seu baixo consumo de memória e por atender a grande parte das necessidades dos *sites* implementados.

De acordo com a pesquisa sobre servidores *Web* feito pela Netcraft em outubro de 2015 (Netcraft, 2015), o NGINX mostrou-se como o segundo servidor *Web* mais popular dentre os *sites* ativos na *Web* pesquisados, sendo utilizado por 15,33% destes. Além disso a pesquisa informou que o NGINX é o mais popular dentre os *sites* mais utilizados da Internet, com 23.66% dos sites utilizando esse serviço como base.

2.3 ANGULARJS

AngularJS é um *framework* feito em JavaScript, que tem como base auxiliar na execução de *single page applications,* que consistem em apenas uma página *HTML* e faz requisições assíncronas ao servidor para buscar mais informações. Foi desenvolvido visando as premissas do padrão de projeto *model-view-view-model*.

O AngularJS lê o código fonte da página que contém as tags especiais e então executa a respectiva diretiva, e faz a ligação entre a *View* e seu *Model*, representado por variáveis JavaScript. Sua utilização é amplamente recomendada para páginas HTML simples que trabalham com requisições a API, sendo *Get* ou *Post*.

2.4 API

A API deste projeto foi implementada através do *micro-framework* Lumen, o Lumen pode ser visto como um fragmento do *framework* Laravel, ambos desenvolvidos pelo mesmo desenvolvedor, Taylor Otwell (LARAVEL, 2017).

O objetivo da criação do Lumen foi pela necessidade de se criar um *framework* para atender uma demanda que necessite de um rápido tempo de resposta para seu funcionamento, focando-se principalmente em componentes básicos e necessários. Como este trabalho é a simulação de um ISP.

Nesta aplicação, a API foi implementada seguindo o padrão REST e responde somente a requisições *Get* ou *Post*, fazendo consultas ao banco de dados e respondendo tais requisições com um objeto *JSON*.

2.5 BANCO DE DADOS

O MySQL é um sistema de gerenciamento de banco de dados de código aberto utilizado na maioria das aplicações gratuitas. O MySQL utiliza linguagem SQL para gerenciar seu conteúdo.Neste projeto o MySQL existe apenas para guardar a descrição do usuário e do conteúdo multimídia. A Figura 3 é a representação gráfica de como foi modelado o Banco de Dados para esse projeto.

Foram apenas três tabelas, com a maioria de suas colunas com nomes autoexplicativos. A partir dessa modelagem, é concedido acesso ao usuário de fazer *upload* de um conteúdo multimídia para o servidor, por isso as tabelas ‘Vídeo e Música’ tem uma chave estrangeira referenciando o usuário.

Na tabela ‘Vídeo’ e na tabela ‘Música’, criou-se uma coluna chamada “caminho”, que representa a localização de onde o arquivo se encontra no servidor, por exemplo, “http://10.0.0.1/usuario123/videos/aniversario.mp4”.

Na tabela ‘Usuário’, a coluna “chave_transmissao” é uma *string* única que cada usuário recebe ao se cadastrar, que representa a chave do endereço RTMP que o usuário irá ou não utilizar para fazer alguma transmissão ao vivo, por exemplo, “rtmp://10.0.0.1/live/chave_transmissao”.

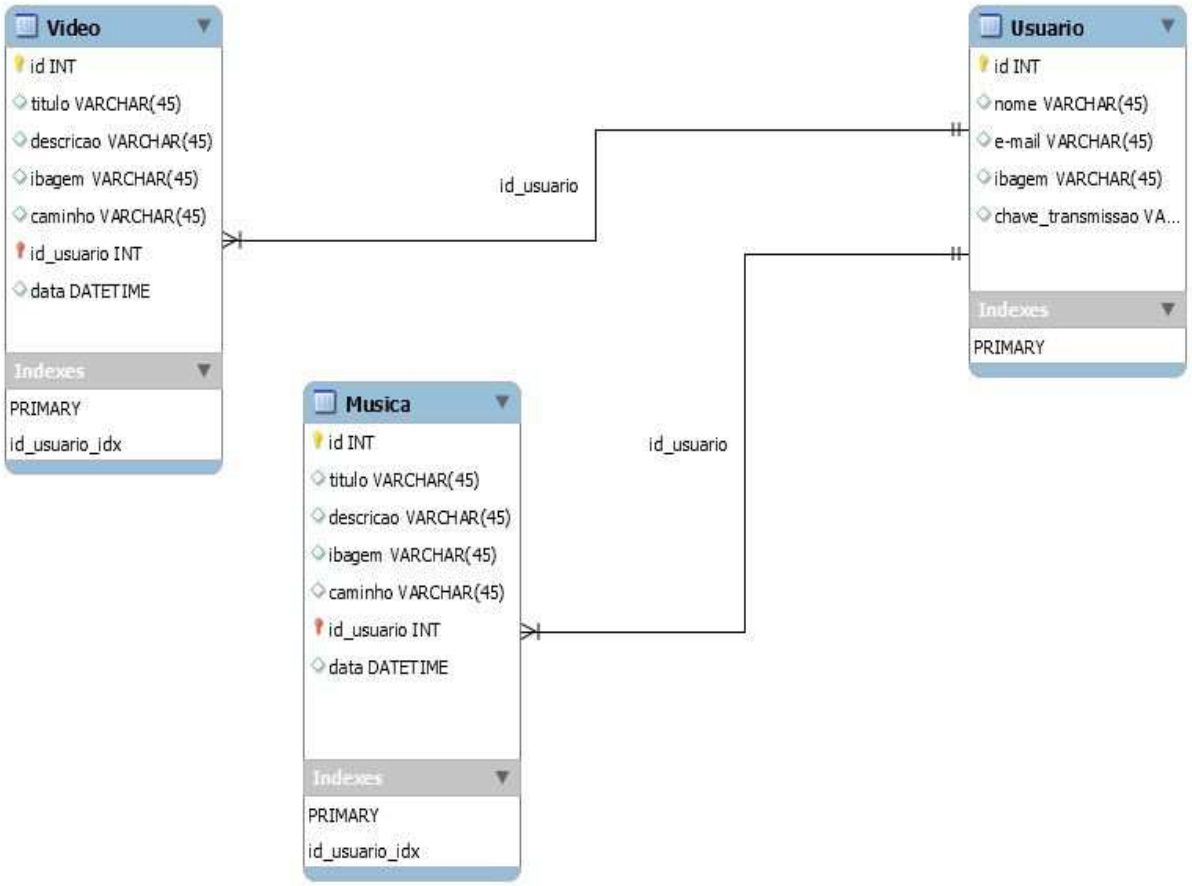

Figura 3: Diagrama EER do MySql

3

IMPLEMENTAÇÃO

O projeto foi desenvolvido a partir da instalação do Ubuntu 14.04 e seguido da instalação de pacotes extras, como o RP-PPPoE e o NGINX.

O RP-PPPoE requer uma pequena configuração para que se possa conceder acesso aos clientes que estarão conectados, configurações de autenticação, implementado através do protocolo CHAP, DNS e escolha de IP padrão. Após isso, foi preciso compartilhar a conexão com a Internet, isso foi feito através do encaminhamento de pacotes entre as interfaces de rede, no caso, do Wifi para a Ethernet.

As configurações utilizadas pelo NGINX são ainda mais simples, bastou habilitar o uso do PHP e instalar o módulo RTMP.

Com o módulo RTMP instalado, foi necessário editar um arquivo de texto para configurar o módulo RTMP. Foi preciso apenas abrir a porta TCP 1935 (porta padrão do RTMP), tamanho do Chunk, habilitar o *publish* de *Live Streams* e especificar um diretório que armazenará todo o conteúdo multimídia.

Para questões de armazenamento de informações, bastou utilizar um SGBD simples, como o MySql.

Para acesso ao conteúdo por parte do cliente, foi apenas preciso implementar uma página HTML e adicionar um *player* a esta página.

3.1 BACK END

Back End é toda a parte que fica do lado do servidor, tal como banco de dados, conteúdo multimídia, implementações de código em linguagens como o PHP. Existem inúmeras linguagens *back end* e utilizou-se o PHP nesse projeto.

Um modo fácil de combinar a comunicação entre *back end* e *front end*, é por meio de uma *API* que responde aos *endpoints,* a API desse projeto foi implementada

apenas para fazer consultas ao banco de dados e retornar a *query* em formato *Json* (objeto Javascript).

3.2 **FRONT END**

Front End é toda a parte que fica do lado do cliente em uma relação cliente-servidor, ou seja, tudo que está rodando no *hardware* do cliente, tal como um código HTML, CSS, Javascript.

Nessa aplicação, o *Front End* fica encarregado de realizar requisições assíncronas *Get* e *Post*, recebendo como resposta do *Back End*, um objeto *Javascript* para ser interpretado de maneira adequada. Esse objeto deve vir com todas as informações necessárias para a reprodução do vídeo ou do *Stream*.

Utilizou-se o AngularJS para uma melhor comunicação de forma assíncrona entre cliente-servidor e para a reprodução do conteúdo foi utilizado o player de vídeo JWPlayer.

A Figura 4 representa o funcionamento do *Back-End* e *Front-End*.

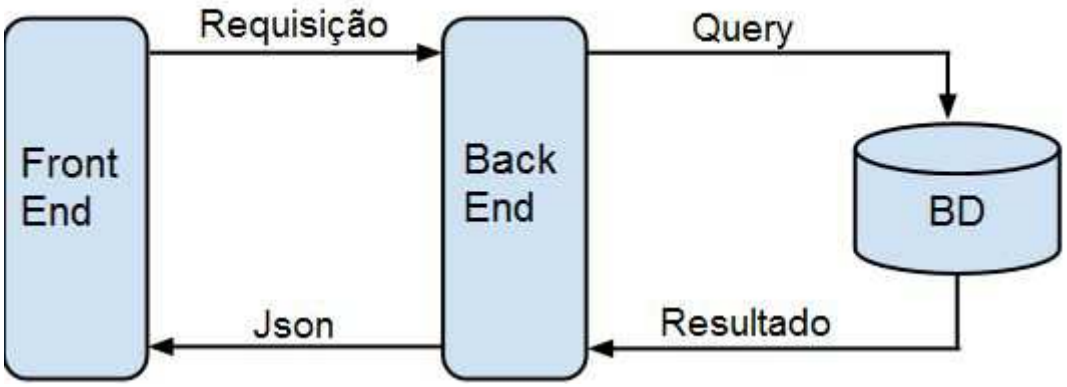

Figura 4: Representação gráfica da comunicação.

4 EXPERIMENTOS

A partir do momento em que todo o ambiente foi configurado e toda a implementação foi finalizada, as avaliações puderam ser efetuadas.

De acordo com o ambiente limitado e restrito a apenas uma conexão *PPPoE,* o objetivo dos experimentos foi de monitorar mudanças que indicassem uma perda relevante na qualidade do serviço do *ISP,* justamente para avaliar esta solução como forma de auxílio para possíveis problemas recorrentes apresentados com o uso do protocolo. A partir desse monitoramento, montou-se uma tabela com os seguintes parâmetros para os experimentos:

- Dispositivos: Número de aparelhos conectados ao ISP.
- Tamanho total: Caso seja a visualização de um conteúdo multimídia armazenado no servidor, esse campo representa a soma de todos os arquivos requisitados por todos os dispositivos conectados, a unidade de medida foi em GB.
- *Live Streams*: Número de *live streams* ativas para visualização.

- *Download*: Taxa de *download* do servidor naquele momento, unidade de medida em MB/s.
- *Upload*: Taxa de *upload* do servidor naquele momento, unidade de medida em MB/s.
- Uso da UCP: Porcentagem do uso da UCP do servidor.

Da mesma forma, com o mesmo monitoramento, montou-se duas tabelas e seus respectivos gráficos com o objetivo de avaliar a qualidade de serviço da aplicação. Um fator importante para medir a qualidade de serviço é o atraso no recebimento de pacotes vindo do servidor (*ping)*, este fator foi usado como parâmetro para a construção destas tabelas. Uma tabela é referente ao conteúdo armazenado no servidor e a outra ao conteúdo transmitido em tempo real (*streaming*).

4.1 CAPTURA DOS DADOS DOS EXPERIMENTOS

Devido ao limite de dispositivos, a primeira e a segunda captura dos dados dos experimentos foram realizadas simulando apenas um único ambiente residencial com dois *smartphones e* um computador *desktop.* A primeira captura é referente a um conteúdo transmitido em tempo real (*streaming*), enquanto a segunda condiz a um conteúdo armazenado no servidor.

Ao término deste experimento, ficou nítido que a estrutura utilizada para este projeto tende a limitar a velocidade, notadamente nos casos que muitos usuários acessam ao mesmo tempo o servidor. Apenas 3 dispositivos conectados exigiram uma banda de 2 MB/s de *Upload* do servidor no caso de *streaming*, levando em conta que o roteador conectado ao servidor tem um limite de tráfego de 18.75MB/s.

A terceira e a quarta fase de experimentos também foram baseada em um único ambiente residencial e utilizou um número maior de dispositivos, nela

foram utilizadas 4 *smartphones, 1 notebook*, 1 *desktop* e 1 *Xbox One*, seguindo o mesmo padrão correspondente as duas primeiras capturas, a terceira captura se baseia em dois conteúdos transmitidos em tempo real (*streaming*), enquanto a quarta se refere a um conteúdo armazenado no servidor.

Os experimentos desta fase reforçam a ideia inicial de que em algum momento a estrutura de redes tenderá a limitar a velocidade na qual cada cliente recebe os pacotes, visto que a taxa de *upload* cresceu muito em especial no caso de *streaming,* isso se reflete no atraso da visualização e/ou prejuízo na qualidade do conteúdo que será apresentado.

A quinta e a sexta fase de experimentos contaram com 12 dispositivos, 6 *smartphones*, 2 *notebooks*, 2 *desktops* e 1 *Xbox one*, e ficou evidente que a largura de banda tende a impactar nesta aplicação, além da largura de banda, nesta captura de experimentos, o ping entre *ISP* e cliente começa a se alterar de forma inesperada em alguns momentos e no caso de *streaming* o uso da UCP cresceu excessivamente.

Tabela 2: Tomada de experimentos

Dispositivos	Tamanho total	Live Streams	Download	Upload	Uso da UCP
3	0 GB	1	0.6 MB/s	0.3 MB/s	5%
3	2.4 GB	0	0.033 MB/s	2 MB/s	4%
7	0 GB	2	1.8 MB/s	0.73 MB/s	9%
7	4.34 GB	0	1.24 MB/s	5.83 MB/s	8%
12	2.4 GB	0	0.11 MB/s	8.0 MB/s	11%
12	0 GB	1	1.78 MB/s	1.8 MB/s	30%

4.2 QUALIDADE DE SERVIÇO EM CONTEÚDO ARMAZENADO NO SERVIDOR

Nos experimentos com vídeos armazenados no servidor notou-se que a perda de pacotes foi nula no ambiente avaliado, porém, outra variável importante de qualidade de serviço foi afetada. O tempo medido para o *ping* cresceu de forma expressiva, este aumento acarretou numa grande redução de performance em aplicações em tempo real paralelas na rede, por exemplo, jogos *online*, em alguns casos a alteração do *ping* inviabiliza a utilização de tais aplicações, a tabela a seguir representa essa análise.

Tabela 3: Monitoramento de atrasos em Conteúdo Armazenado

Dispositivos	Menor atraso	Atraso médio	Maior atraso
3	0 ms	0 ms	5 ms
7	0 ms	0 ms	9 ms
12	0 ms	1 ms	15 ms

O gráfico a seguir demonstra o aumento gradativo no atraso do recebimento dos pacotes. Cada ponto no gráfico é medido em milissegundos e cada cor representa o número de dispositivos conectados na hora da medição.

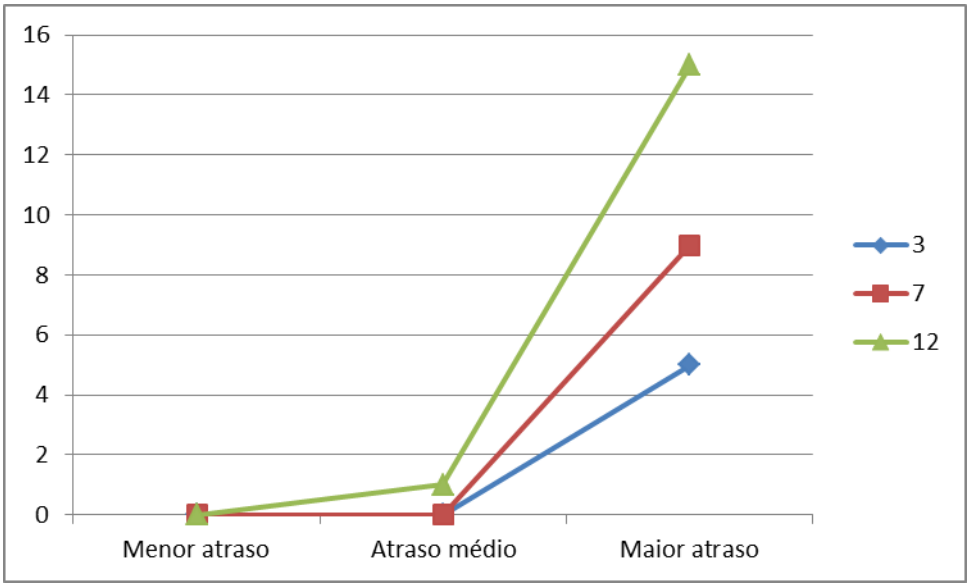

Gráfico 1: Monitoramento de atrasos em Conteúdo Armazenado

4.3 QUALIDADE DE SERVIÇO EM STREAMING

Nos experimentos com vídeos por *streaming* o resultado da perda de pacotes também foi zero no ambiente avaliado. O *ping* seguiu um padrão parecido com os experimentos anteriores, tendendo a aumentar de forma expressiva. O uso de processamento do servidor cresceu notadamente de acordo com a quantidade de dispositivos consumindo o conteúdo, sendo visível que quanto maior for o número de clientes consumindo um conteúdo por *streaming* maior será o uso da capacidade de processamento do servidor.

Tabela 4: Monitoramento de atrasos no *Streaming*

Dispositivos	Menor atraso	Atraso médio	Maior atraso	Uso da UCP
3	0 ms	0 ms	4 ms	4%
7	0 ms	1 ms	5 ms	9%
12	0 ms	1 ms	9 ms	30%

O gráfico 2 demonstra o aumento gradativo no atraso do recebimento dos pacotes, cada ponto no gráfico é medido em milissegundos e cada cor representa o número de dispositivos conectados no momento da medição.

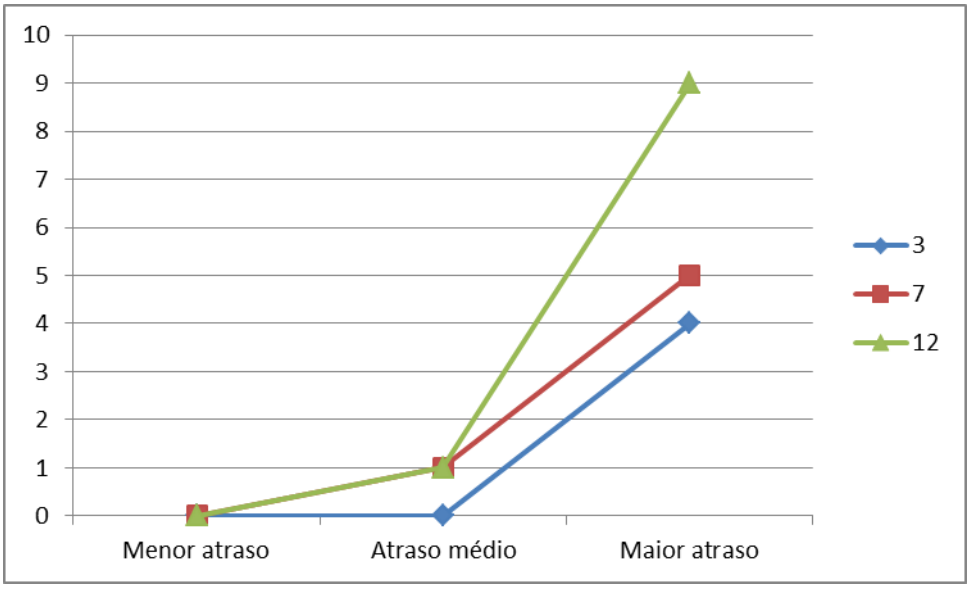

Gráfico 2: Monitoramento de atrasos no *Streaming*

O gráfico 3 demonstra o aumento no uso da UCP, cada ponto no gráfico é representado em milissegundos e cada cor representa o número de dispositivos conectados no momento da medição, este número de dispositivos está relacionado com os experimentos (número 3 no gráfico está relacionado com o experimento 1, número 7 com o experimento 2, e assim por diante).

Pode-se observar que o crescimento não é diretamente proporcional ao número de clientes, Note que do segundo para o terceiro experimento não foram duplicados os dispositivos e o uso da UCP mais que triplicou,crescendo de

aproximadamente 9% para 30%. Portanto, isso indica que o planejamento e gerenciamento da capacidade, não só de rede, mas também de processamento são fatores relevantes para esse tipo de solução.

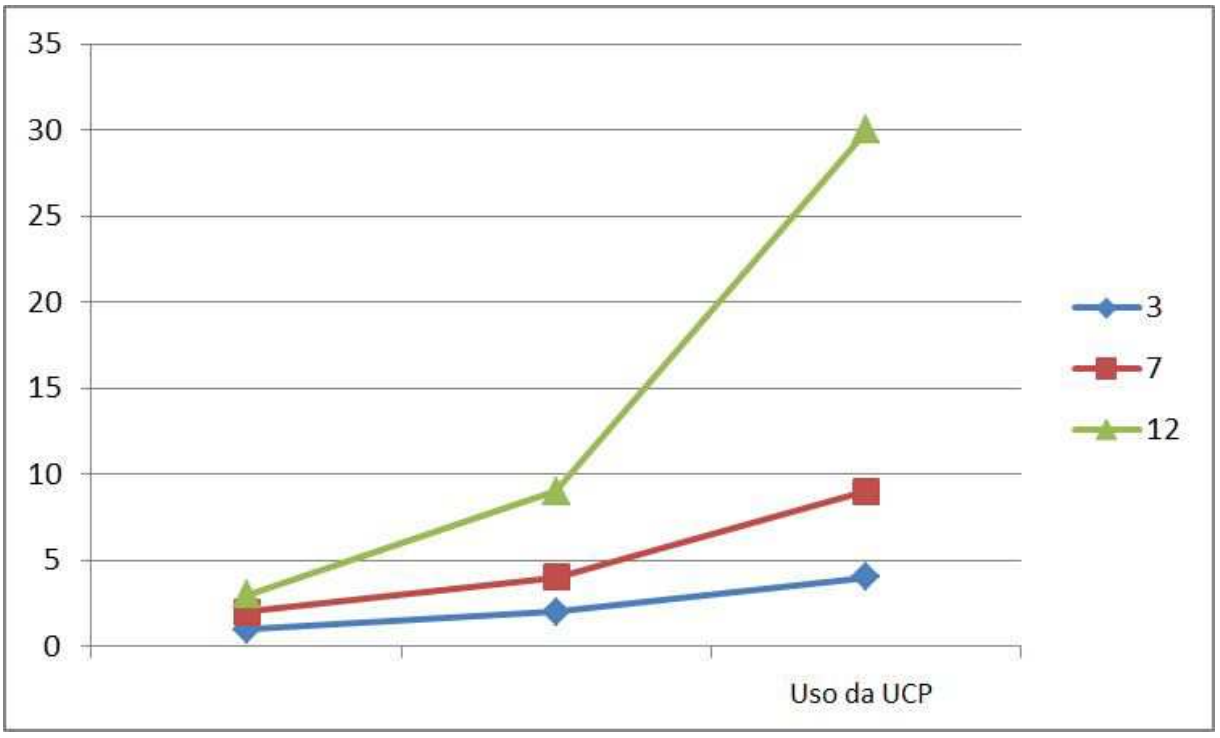

Gráfico 3: Monitoramento do Uso da UCP no *Streaming*

5

CONCLUSÕES

Como o ambiente utilizado neste projeto mostrou-se bastante limitado, os testes ficaram restritos a somente uma conexão PPPOE. No entanto, é perfeitamente normal que uma única conexão PPPOE tenha diversos dispositivos conectados, assim como acontece numa residência, onde temos computadores, celulares e outros dispositivos com acesso à rede.

Dentro do esperado, conseguiu-se acesso ao conteúdo multimídia de forma rápida e bastante estável em todos os dispositivos, tanto computadores quanto celulares. E além do conteúdo sob demanda e vídeos em extensão mp4 armazenados no servidor, foi possível obter acesso a todas as *Live Streams* ativas naquele momento, bastando para isso fazer com que um dispositivo publique sua *Live Stream* para o servidor NGINX, para *Smartphones Android*, foi utilizado o aplicativo RTMP Camera e para *desktops*, o *software Open Broadcaster Software*.

Nos experimentos com vídeos armazenados no servidor notou-se que uma variável importante de qualidade de serviço foi afetada, o *ping*, que por sua vez foi incrementado de forma expressiva, Este aumento acarretaria em uma grande perda de performance por parte de alguma aplicacão em tempo real que estivesse sendo executada de forma paralela na rede, por exemplo, jogos *online*, em alguns casos a alteração de *ping* inviabiliza a utilização de tais aplicações.

Nos experimentos com vídeos exibidos em tempo real (*streaming*) o *ping* seguiu um padrão semelhante ao teste anterior, tendendo a aumentar de forma expressiva. O uso de processamento no servidor cresceu de forma mais acentuada que a quantidade de dispositivos consumindo o conteúdo. Dessa forma, quanto maior for o número de clientes consumindo um conteúdo por *streaming* maior será o uso da capacidade de processamento do servidor, cabendo um planejamento da capacidade instalada para atender adequadamente aos usuários.

Além disso, existe uma concreta viabilidade de uma transmissão ao vivo para o *ISP* através dos mecanismos mostrados nesse projeto. Isso acarretaria em uma

transmissão ao vivo e não necessitaria do uso do receptor ou de antenas na casa do cliente, apenas uma *Smart Tv* conectada ao ISP.

Essa lógica é parecida com a do site *twitch.tv* usa, no caso do *twitch*, existe um aplicativo próprio para *Smartphones* e *Smart Tvs* que possibilita a visualização das *Lives Streams*, faltando apenas concentrar tal conteúdo em um *ISP*.

Para aplicações futuras e principalmente contando com possibilidades de grandes investimentos, seria possível implementar esse projeto e atender uma maior demanda de dispositivos conectados. através do estudo realizado nesse projeto verificou-se ser perfeitamente viável atender esta demanda de usuários com a condição de que o poder computacional e os serviços de rede sejam adequados a esta demanda e escaláveis.

Uma das possibilidades de aperfeiçoamento visando o cenário brasileiro e usuários dependentes de aplicações em tempo real seria a utilização de um segundo servidor dedicado ao processamento do conteúdo a ser exibido por *streaming,* isto é, *live streaming* e multimídia sob demanda, Este incremento resultaria em um ISP que não desperdiça poder computacional com o processamento do conteúdo multimidia que será requisitado pelos clientes, uma vez em que tal *streaming* está sendo processado em outra máquina.

A principal desvantagem dessa arquitetura seria um maior custo de manutenção de um servidor extra, mas levando em consideração aplicações onde a principal experiência do usuário seria o consumo de conteúdo por *streaming* e que a tendência desse cenário é de crescimento, o investimento em outro servidor tende a ser indispensável e economicamente viável.

6

REFERÊNCIAS BIBLIOGRÁFICAS

1. ADOBE, Real-Time Messaging Protocol (RTMP) specification. Disponível em <http://www.adobe.com/devnet/rtmp.html>. Acesso em: 15 de dezembro de 2016
2. GOOGLE, AngularJS. Disponível em <https://angularjs.org>. Acesso em 4 de janeiro de 2017.
3. H. Parmar, Ed.; M. Thornburgh, Ed. *Adobe's Real Time Messaging Protocol Specification.* Disponível em <http://www.adobe.com/devnet/rtmp.html>. Acesso em 15 de janeiro de 2017.
4. JWPLAYER. Disponível em <https://www.jwplayer.com/>. Acesso em 4 de janeiro de 2017
5. LARAVEL, Lumen. Disponível em < https://lumen.laravel.com>. Acesso em 4 de janeiro de 2017.
6. Netcraft. Disponível em < https://news.netcraft.com/archives/2015/10/16/october-2015-web-server-survey.html >. Acesso em: 9 de abril de 2017.
7. NGINX INC, nginx about. Disponível em < https://nginx.org/en>. Acesso em: 3 de janeiro de 2017
8. ORACLE, MySql. Disponível em < https://www.mysql.com/>. Acesso em 26 de março de 2017.
9. THE PHP GROUP, O que é PHP? Disponível em < https://secure.php.net/manual/pt_BR/intro-whatis.php>. Acesso em: 4 de janeiro de 2017.
10. TILKOV, Stepan. Uma rápida Introdução ao REST, 2008. Disponível em:< https://www.infoq.com/br/articles/rest-introduction> Acesso em: 3 de janeiro de 2017.