UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

BRENO MAGALLINI LEITE

JOGOS SOB DEMANDA: IMPLEMENTAÇÃO DE UM SERVIDOR *STREAMING*

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2016

BRENO MAGALLINI LEITE

JOGOS SOB DEMANDA: IMPLEMENTAÇÃO DE UM SERVIDOR *STREAMING*

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação do Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná.

Orientador: Dr. André Koscianski

PONTA GROSSA

2016

TERMO DE APROVAÇÃO

JOGOS SOB DEMANDA: IMPLEMENTAÇÃO DE UM SERVIDOR *STREAMING*

por

BRENO MAGALLINI LEITE

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 18 de novembro de 2016 como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Dr. André Koscianski Prof. Orientador	Dr. Richard Duarte Ribeiro Membro titular
	Dr. Gleifer Vaz Alves Membro titular
Dr. Augusto Foronda Responsável pelos Trabalhos de Conclusão de Curso	Dr. Erikson Freitas De Morais Coordenador do Curso UTFPR – Campus Ponta Grossa

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

Dedico este trabalho aos meus pais, pelo apoio e motivação.

AGRADECIMENTOS

Primeiramente agradeço aos meus pais, Marilene e Paulo, por todo o apoio que me deram durante toda a minha vida. Sem eles eu seria nada, durante todo esse tempo eles foram a principal razão de todo meu esforço em me tornar alguém melhor.

Queria também agradecer a minha namorada, Ana Carolina, que durante todo este tempo, mesmo estando longe, se fazia perto para todos os momentos em que eu precisei.

Agradeço aos meus amigos Diogo, Bruno, Jonas e Lucas Migliorini por todo o apoio, noites em claro e momentos que passamos juntos. Queria que soubessem que sem vocês a faculdade não teria sido a mesma. Também agradeço a todos meus outros amigos que tive a honra de conhecer nessa etapa da minha vida.

Um agradecimento em especial ao meu orientador, André Koscianski. Por ter me orientado não só neste trabalho, mas por ter me ensinado coisas que levarei para a vida toda.

Também sou eternamente grato à professora Sheila, que com seus ensinamentos e conversas me cativou e me mostrou o tipo de pessoa que quero ser no futuro. Agradeço também aos professores Gleifer e Saulo, sem vocês nossa formação não seria tão boa quanto foi.

E por último mas não menos importante, queria agradecer a todos os professores que de certa forma contribuíram com a minha formação, um grande obrigado.

RESUMO

LEITE, Breno Magallini. **Jogos sob demanda:** implementação de um servidor *streaming*: 2016. 51 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2016.

Os jogos normalmente necessitam de uma alta demanda de *hardware*, o que torna milhares de dispositivos inaptos para processá-los. Este trabalho apresenta um sistema que pode tornar essa realidade um pouco diferente. Jogos sob demanda é o nome dado a esse sistema, o mesmo se baseia em computação distribuída e *streaming* de vídeo. Tem como intuito transferir o custo computacional de processar um jogo do computador do usuário para um servidor, que é preparado para lidar com o alto nível de processamento. Neste trabalho este sistema foi desenvolvido utilizando as bibliotecas *SDL* e *zlib,* com objetivos de renderização e compressão dos *frames,* respectivamente. Este sistema foi testado utilizando diversas redes e diversas resoluções para o jogo, com intuito de demonstrar as diferenças obtidas nas taxas de *frames* por segundo dependendo do ambiente. Utilizando-se de baixas resoluções o sistema obteve uma taxa de 35 *frames* por segundo, conforme as resoluções eram aumentadas a taxa de *frames* era reduzida proporcionalmente. O trabalho também propõe algumas soluções possíveis para essa redução na taxa dos *frames* como trabalhos futuros.

Palavras-chave: Jogos. Jogos Sob Demanda. Jogos Distribuídos. *Streaming* de Vídeo.

ABSTRACT

Leite, Breno Magallini. **Games on demand:** implementation of a streaming video server: 2016. 51 p. Work of Conclusion Course (Bachelor of Computer Science) - Federal Technology University - Paraná. Ponta Grossa, 2016.

Games typically require a high hardware demand, which makes a thousands of devices incapable to process them. This paper presents a system that can make this reality a little different. Games on demand is the name given to this system, It relies on distributed computing and video streaming. The objective of the system is to transfer the computational cost of a game from the user's computer to a server, which is prepared to the high processing demand. In this paper, this system was developed using the SDL and zlib libraries. The former has the objective of rendering the game frames, the latter is responsible to compress the frames. The system was tested using several networks and several resolutions in order to show the differences obtained in the frame rate depending on the environment. Using low resolutions, The system obtained a frame rate of 35 frames per second. However as resolutions were increased the rate was reduced proportionately. This paper also proposes some possible solutions for this reduction in the frame rate as future works.

Keywords: Games. Games on Demand. Distributed Games. Video Streaming.

LISTA DE ILUSTRAÇÕES

Figura 1 – Funcionamento de um jogo. 17
Figura 2 – Perspectivas do usuário com diferentes taxas de FPS. 18
Figura 3 – Exemplo de uma arquitetura cliente/servidor. 19
Figura 4 – Divisão do jogo na arquitetura cliente/servidor. 24
Figura 5 – Ilustração de um *gameloop*. 25
Figura 6 – Funcionamento detalhado do sistema. 25
Figura 7 – Ilustração do jogo. 27
Figura 8 – Fluxograma do servidor. 28
Figura 9 – Fluxograma do cliente. 30
Figura 10 – Arquitetura do sistema. 32
Figura 11 – Comparativo de tamanho/qualidade da compressão JPEG. 46
Figura 12 – Funcionamento da compressão MPEG. 47
Figura 13 – Comparação do vídeo original e no formato *.MP4* 48

Quadro 1 – Dispositivos utilizados para teste 34
Quadro 2 – Variação nas taxas de FPS obtidas nos testes. 39

Gráfico 1 – Comparativo do crescimento do tamanho dos *frames* com e sem compressão. 35
Gráfico 2 – Comparando tamanho nos formatos *.MP4* e *.MJPEG* 49

LISTA DE TABELAS

Tabela 1 – Comparação de *frames* com e sem compressão. 35
Tabela 2 – Comparativo do tempo médio de compressão e descompressão de um *frame*. 36
Tabela 3 – Comparativo do tempo médio de compressão e descompressão de um *frame* com resolução 600x400 utilizando diferentes dispositivos. 36
Tabela 4 – Média de tempo de envio dos *frames*. 38

LISTA DE ABREVIATURAS E SIGLAS

CPU	*Computer Processing Unit*
FPS	*Frames per Second*
GPU	*Graphics Processing Unit*
I/O	*Input/Output*
IP	*Internet Protocol*
MJPEG	*Motion JPEG*
MPEG	*Moving Picture Experts Group*
RTP	*Real Time Protocol*
RTSP	*Real Time Stream Protocol*
SDL	*Simple DirectMedia Layer*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*

LISTA DE SÍMBOLOS

GHz	*Gigahertz*
Mbps	*Megabits per seconds*
MB	*Megabytes*
ms	*Milliseconds*
μ	*Microseconds*
kB	*Kilobytes*

SUMÁRIO

1 INTRODUÇÃO **12**
1.1 OBJETIVOS 13
1.1.1 Objetivo Geral 13
1.1.2 Objetivos Específicos 13
1.2 JUSTIFICATIVA 13
1.3 ORGANIZAÇÃO DO TRABALHO 14
2 FUNDAMENTAÇÃO TEÓRICA **15**
2.1 TRABALHOS RELACIONADOS 15
2.1.1 Nvidia - *Shield/Geforce Now* 15
2.1.2 Sony - *Playstation Now* 16
2.1.3 *Gaminganywhere* 16
2.2 FUNCIONAMENTO DOS JOGOS 17
2.3 COMPUTAÇÃO DISTRIBUíDA 18
2.3.1 Arquitetura Cliente/Servidor 19
2.4 PROTOCOLOS DE REDE 20
2.4.1 Protocolo TCP 20
2.4.2 Protocolo UDP 21
2.4.3 Protocolos RTP e RTSP 21
2.5 *STREAMING* DE VíDEO 22
2.5.1 *Codec* de Vídeo 23
2.6 JOGOS SOB DEMANDA 24
3 DESENVOLVIMENTO **27**
3.1 O JOGO UTILIZADO 27
3.2 SERVIDOR 28
3.3 O CLIENTE 29
3.4 COMUNICAÇÃO ENTRE SERVIDOR E CLIENTE 31
3.5 ARQUITETURA DO SISTEMA 32
4 RESULTADOS **34**
4.1 DISPOSITIVOS UTILIZADOS NOS TESTES 34
4.2 COMPRESSÃO 34
4.3 TEMPO DE ENVIO DOS *FRAMES* 37
4.4 TAXA DE FPS 39
5 CONCLUSÃO **40**
5.1 TRABALHOS FUTUROS 40
REFERÊNCIAS **42**
APÊNDICE A - Comparativo entre MJPEG e MPEG **45**

1 INTRODUÇÃO

A computação em nuvens é um termo que vem crescendo nos últimos anos. Segundo Taurion (2009) a computação em nuvens será o futuro da Internet. Devido ao crescimento dessa tecnologia algumas companhias como Netflix (2015), Spotify (2015) e Google (2015) já disponibilizam serviços que utilizam-se desses conceitos. Conceitualmente, computação em nuvens é o termo dado a qualquer serviço, seja ele de *software* ou *hardware*, disponibilizado via Internet (ARMBRUST *et al.*, 2009).

Este conceito já existe na nossa sociedade atual. Já não é necessário sair de casa para assistir a um filme, por exemplo com plataformas como o *Netflix* pode-se acessar centenas de filmes em questão de segundos. Pode-se imaginar que o mesmo fosse possível para os jogos digitais, evitando-se horas baixando e instalando jogos, ou ainda que as máquinas não precisassem ser super potentes para jogar os jogos mais recentes e que não fosse preciso atualizar as máquinas todos os anos em razão do rápido avanço da tecnologia. Essa é uma realidade que já existe e ela é chamada de jogos sob demanda ou simplesmente jogos em nuvens.

Jogos sob demanda é o termo dado a um sistema cliente/servidor que proporciona aos jogadores a experiência de jogar algo que está sendo executados em um servidor e não no computador ou dispositivo do usuário. Em jogos sob demanda a execução do jogo é dividida em duas etapas: o cliente é responsável por receber as ações do jogador e de mostrar os gráficos do jogo para o usuário. Já o servidor se encarrega do processamento de informação, ou seja da criação dos gráficos, do cálculo da física, da lógica envolvida no jogo e da aplicação da inteligência artificial dentro do jogo (BARBOZA *et al.*, 2010).

Essa divisão proporciona uma boa qualidade dos gráficos com um baixo custo de *hardware* para o cliente, pois o processamento é transferido para o servidor tornando baixa a necessidade de *hardware* no cliente. Desta forma, os sistemas de jogos sob demanda tornam os jogos mais acessíveis para os usuários pois qualquer pessoa que tenha um dispositivo eletrônico, seja ele um *smartphone*, notebook ou computador, pode ter acesso a diversos jogos utilizando-se da computação distribuída.

Porém, existem inúmeras dificuldades que precisam ser superadas para que esses sistemas possam se tornar uma realidade para todos. Dentre esses problemas estão: formas de comprimir e capturar os *frames* e a alta necessidade de uma conexão eficaz com a Internet. Este trabalho mostra algumas formas de solucionar esses problemas, com o intuito de provar que a criação de um sistema de jogos sob demanda é possível com as tecnologias atuais.

1.1 OBJETIVOS

Esta seção apresenta o objetivo geral e os objetivos específicos que norteiam o desenvolvimento deste trabalho.

1.1.1 Objetivo Geral

Desenvolver um sistema de jogos sob demanda, onde o processamento do jogo ocorra em um computador robusto (servidor), enquanto o jogo é jogado em um dispositivo mais simples (cliente).

1.1.2 Objetivos Específicos

Para que seja possível o desenvolvimento deste trabalho é necessário realizar os seguintes objetivos específicos:

- Escolher um jogo 2D que sirva de base para o desenvolvimento do sistema;
- Desenvolver um servidor que capture e envie os *frames* capturados do jogo;
- Desenvolver um cliente que mostre os *frames* e controle as ações do jogador;
- Analisar e selecionar o melhor protocolo de rede para solucionar o problema proposto;
- Integrar o cliente e servidor, utilizando o conceito de jogos sob demanda.

1.2 JUSTIFICATIVA

Com o alto crescimento de usuários com acesso à Internet (STATISTA, 2015a) e o crescimento do mercado na área de jogos digitais (STATISTA, 2015b), as grandes empresas da área começaram a dar mais valor para sistemas que possam atingir mais usuários como os sistemas de jogos em nuvens. Uma prova da importância que as grandes empresas estão dando para esta área é a aquisição da empresa OnLive pela Sony (NICK, 2015). A OnLive era uma empresa que disponibilizava um sistema de jogos sob demanda *online*, a mesma foi comprada pela Sony como uma forma de controlar o mercado. Logo após a compra o seu serviço foi descontinuado.

Os jogos sob demanda facilitam vários aspectos na área dos jogos, tanto para os jogadores quanto para os desenvolvedores. Huang *et al.* (2013) descrevem algumas dessas facilidades, sendo as principais: os dispositivos se tornam menos obsoletos, ou seja, o sistema reduz necessidade de melhorias contínuas nos computadores e dispositivos; O sistema permite que qualquer jogo seja multiplataforma, o que torna a criação de uma única versão do jogo mais viável para os desenvolvedores, facilitando e reduzindo custos na hora do desenvolvimento.

Grandes empresas como Google e Apple já possuem serviços disponíveis que utilizam do conceito de computação em nuvens, alguns deles são o Google Drive, Docs, Sheets e o iCloud. Parece existir uma tendência que no futuro cada vez mais empresas disponibilizem serviços *online* e que os computadores precisem de uma quantidade gradativamente menor de *hardware*, devido a grande parte do processamento ser feito nas nuvens.

1.3 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está organizado da seguinte forma.

O Capítulo 2, apresenta uma visão geral sobre o funcionamento do sistema proposto neste trabalho, assim como os conhecimentos básicos necessários para o desenvolvimento do mesmo. Dentre os assuntos nele tratados estão sistemas distribuídos, arquitetura cliente e servidor, como funcionam os jogos e o funcionamento geral de um sistema de jogos sob demanda.

O Capítulo 3, descreve os procedimentos realizados para a implementação do sistema neste trabalho proposto, assim como linguagens e tecnologias utilizadas. Nesse capítulo também são discutidas as melhores formas de se implementar esse sistema, assim como os problemas encontrados durante o desenvolvimento do mesmo.

O Capítulo 4, apresenta o desempenho obtido pelo sistema implementado utilizando de métricas como taxa de FPS e tempo de resposta.

Por fim, o Capítulo 5, descreve as conclusões obtidas neste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Para que seja possível o entendimento do sistema a ser criado neste trabalho é necessário um conhecimento básico sobre processamento de jogos além de alguns conceitos de redes que serão utilizados para distribuir o processamento entre os computadores. Também é importante o entendimento do que é um *streaming* de vídeo, pois esta tecnologia será utilizada na resolução do problema proposto. Este capítulo tem como propósito explicar esses conceitos necessários para o desenvolvimento deste trabalho.

2.1 TRABALHOS RELACIONADOS

Esta seção tem como objetivo mostrar trabalhos relacionados ao tema, mais informações sobre eles pode ser encontrados na página de referências.

2.1.1 Nvidia - *Shield/Geforce Now*

A Nvidia foi uma das empresas pioneiras na criação desses sistemas de jogos sob demanda, com o seu sistema chamado *Geforce Now* (NVIDIA, 2015a). Para um auxilio a esta tecnologia foi criada uma placa de vídeo com um enfoque em virtualização de sistemas, essa placa é utilizado para que as GPUs possam fazer um processamento distribuído dos jogos (NVIDIA, 2015b).

O sistema tem o seu código fonte fechado, o que torna impossível a análise do código fonte. Sendo assim, não é conhecido o exato funcionamento da tecnologia, porém analisando a placa de vídeo criada junto com o sistema e o enfoque dado pela mesma a virtualização, pode-se projetar como a tecnologia foi implementada. Devido a análise acredita-se que o sistema funciona capturando as imagens do jogo diretamente da tela do computador, desta maneira para cada jogo que é executado no servidor há uma necessidade de uma nova máquina virtual, mais detalhes sobre esta maneira de obter os *frames* do jogo sera discutida nos próximos capítulos deste trabalho. Esta suposição explicaria o motivo da criação de uma placa de vídeo voltada para virtualização.

A Nvidia criou alguns *consoles* chamados *Shield* com o intuito de serem utilizados como cliente no sistema de jogos sob demanda, desta maneira para usufruir do serviço disponibilizado há necessidade de obter um dos *consoles* da companhia, além de pagar uma quantia por mês. O serviço possui uma biblioteca de jogos relativamente

grande e o serviço tem uma boa qualidade em geral.

2.1.2 Sony - *Playstation Now*

Empresas como a Sony também possuem seu sistema de jogos sob demanda, chamado *Playstation Now* (SONY, 2015). O sistema é bem similar com o sistema da Nvidia, pois é necessário a adesão do *console* da empresa. O sistema, como todo outro produto de empresas privadas, tem o seu código fonte fechado, por esse motivo não é possível dizer exatamente como o sistema funciona. Mas acredita-se que o mesmo também faz a captura dos *frames* diretamente da tela, porém essa é só uma suposição, há a possibilidade de que o mesmo seja feito de outra maneira, como capturando os *frames* diretamente do *buffer* do jogo.

Como a Sony já possuía seus *consoles*, o serviço de jogos sob demanda foi simplesmente adicionado a eles. É interessante perceber que as duas empresas estão apostando em dispositivos móveis como *tablets* para essa nova tecnologia, isso somente é possível pelo motivo de o processamento no cliente ser muito baixo devido a distribuição de processamento da arquitetura cliente/servidor. Um dos motivos do foco nesses dispositivos móveis pelas companhias é o seu grande crescimento em relação aos demais dispositivos (EXAME, 2015).

2.1.3 *Gaminganywhere*

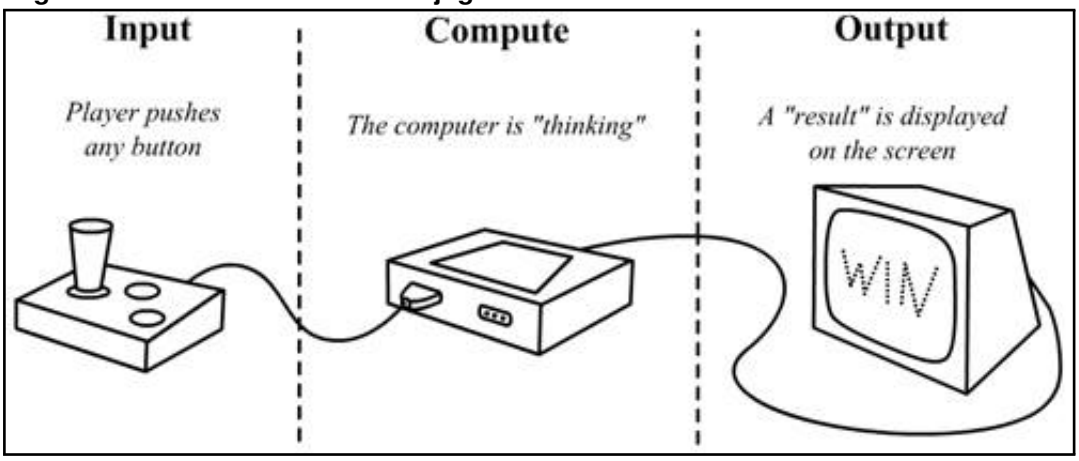
Este é um trabalho criado com enfoque para o estudo de como os sistemas de jogos sob demanda funcionam, o seu código fonte é aberto proporcionando assim uma forma de ver como o mesmo trata todas as informações necessárias no sistema. Nele, os *frames* são capturados da tela do jogo com a necessidade de a criação de uma máquina virtual e/ou escolha de dimensão da tela para executar mais de um jogo ao mesmo tempo (HUANG *et al.*, 2013). Os desenvolvedores deste trabalho deixam claro que o enfoque do mesmo é pesquisa, o trabalho utilizou a linguagem C/C++ tanto para a criação do servidor como do cliente.

Pelo sistema ser voltado para estudos, os desenvolvedores não se importaram em automatizar os processos. Tornando o uso do mesmo difícil para usuários, ou para pessoas que não tenham um conhecimento básico do tema. O sistema precisa de várias configurações como IP das máquinas a serem utilizadas, tamanho da tela do servidor e do cliente, assim como informações sobre as imagens a serem transferidas.

2.2 FUNCIONAMENTO DOS JOGOS

O processamento de um jogo pode ser dividido em duas partes: a primeira é conhecida como I/O (entrada e saída), nesta parte o usuário interage com o jogo, essa interação refere-se tanto as ações que o usuário pode executar no jogo como as imagens que estão sendo mostradas para o jogador. A segunda parte deste processamento, é onde toda a lógica do jogo acontece, nela são aplicados conceitos de física, inteligência artificial entre outros com intuito de gerar os novos cenários do jogo que serão mostrados para o usuário. A Figura 1 mostra graficamente os componentes que envolvem um jogo.

Figura 1 – Funcionamento de um jogo.



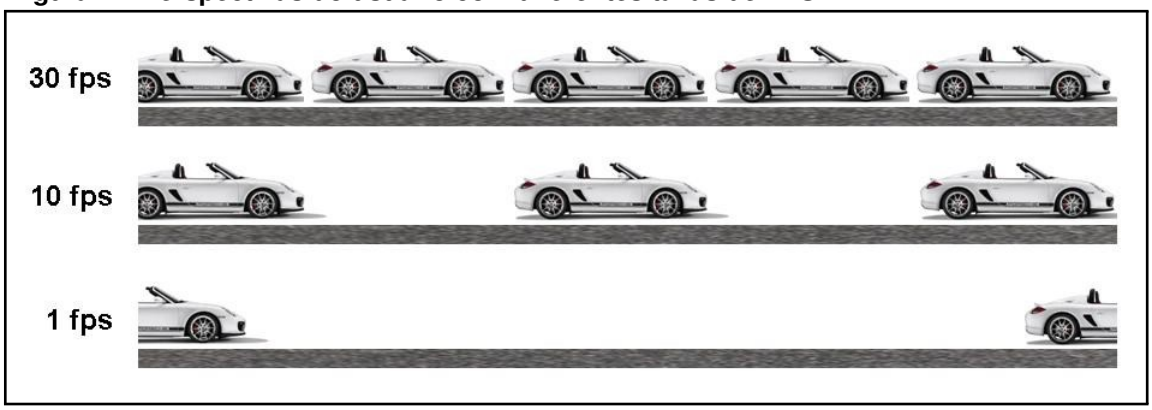
Fonte – Djaouti *et al.* (2008)

A parte lógica (*compute*) do jogo, é geralmente complexa e seu custo computacional é elevado comparado com a parte de entrada e saída (*Input/Output*). Este alto custo é devido a grande quantidade de cálculos que precisam ser executados para gerar os novos *frames* do jogo, *frames* é o nome dado as imagens geradas pelo jogo e são exibidas sequencia para o jogador. Devido a velocidade em que os *frames* são atualizados o jogador tem a impressão de estar vendo uma imagem em movimento.

A quantidade de *frames* exibidos por segundo (Frames Per Second - FPS), é uma das medidas mais utilizadas para analisar o desempenho de um computador executando um jogo. Quanto mais alta a taxa de FPS, mais *frames* o jogo estará gerando e mais suave será a transição entre os *frames*. Uma taxa de FPS que esteja entre 30 e 60 é considerável imperceptível para o olho humano, ou seja, a mudança entre os *frames* é suficientemente rápida para dar a impressão de uma imagem em movimento. Mais informações sobre como o FPS influência na jogabilidade pode ser encontrada em Claypool, Claypool e Damaa (2006). A Figura 2 mostra a perspectiva de um jogador vendo a mesma imagem com diferentes taxas de FPS.

A CPU por si só não possui um poder computacional suficiente para processar

Figura 2 – Perspectivas do usuário com diferentes taxas de FPS.



Fonte – Newton (2013)

toda a informação necessária pela parte lógica do jogo e a criação contínua de todos os *frames* necessários. Com intuito de auxiliar a CPU, foi criado um novo componente que é conhecido como GPU. Ela possui um alto poder computacional dedicado ao processamento gráfico, desta forma, é possível o processamento e a geração dos *frames* em pouco tempo, possibilitando que o usuário não perceba a ocorrência dessa troca. Devido a grande capacidade computacional das GPUs e a necessidade dos jogos em executar cálculos no menor período de tempo possível, as GPUs são amplamente utilizadas para o processamento dos jogos modernos.

2.3 COMPUTAÇÃO DISTRIBUÍDA

Tanenbaum e Steen (2007) descrevem um sistema distribuído como uma coleção de computadores independentes trabalhando de forma que, para os usuários, pareçam um único sistema. Esses computadores podem estar próximos um dos outros ou a centenas de quilômetros conectados por uma rede. Segundo Tanenbaum (2003), uma rede de computadores é um conjunto de computadores autônomos interconectados por uma única tecnologia, dois desses computadores são considerados interconectados quando podem trocar informações entre si. Estes sistemas proporcionaram para a computação uma maneira de superar a capacidade de processamento dos computadores, isso é possível, pois um problema que tem uma alta complexidade pode ser dividido em mais simples, onde cada um deles pode ser resolvidos separadamente em diferentes computadores ao mesmo tempo.

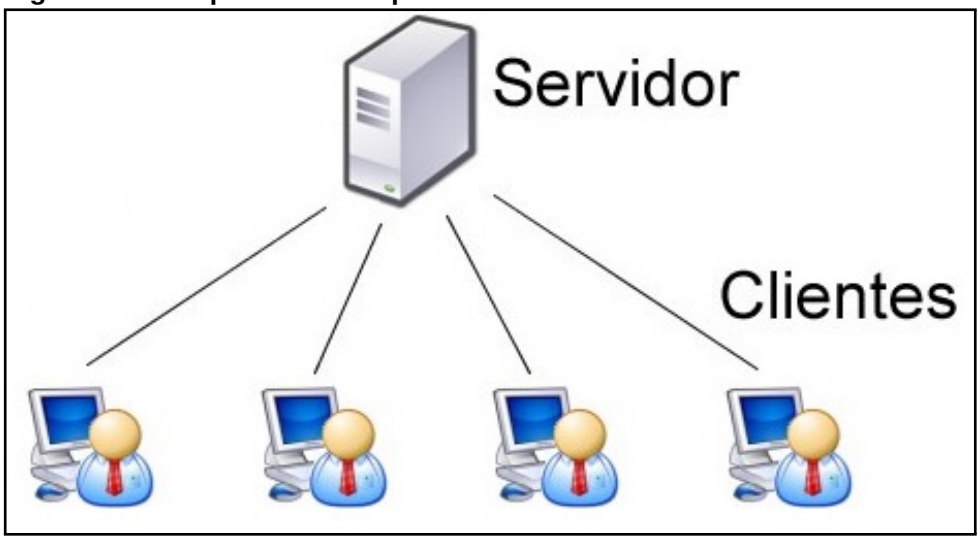
A divisão clara do processamento do jogo em duas partes como mostrado na seção anterior, é um cenário onde a computação distribuída pode ser aplicada. Existem diversos modelos de arquiteturas para sistemas de computação distribuída (COULOURIS *et al.*, 2013). Neste trabalho a arquitetura cliente/servidor será utilizada devido a divisão do problema em duas etapas.

2.3.1 Arquitetura Cliente/Servidor

Arquitetura cliente/servidor é a composição de dois ou mais computadores conectados por uma rede, onde um dos computadores é o fornecedor das informações (servidor) e os outros computadores (clientes) utilizam estas informações para solucionar o problema (BERSON, 1992).

O objetivo do servidor é gerenciar todos os seus clientes de forma que todos executem suas tarefas independentemente uns dos outros, no servidor ocorre o processamento de alto custo proporcionando um menor processamento nos clientes. Devido a essa separação de alto e baixo nível de processamento, o servidor tende a ter uma maior capacidade de processamento que seus clientes. A Figura 3 exemplifica a organização de uma arquitetura cliente/servidor, onde vários clientes estão conectados com um servidor por meio de uma rede qualquer.

Figura 3 – Exemplo de uma arquitetura cliente/servidor.



Fonte – Soriano (2015)

Esses clientes enviam requisições para o servidor que as processa e retorna as respostas, essa troca de mensagens entre o cliente e o servidor pode ser feita de várias maneiras. É importante que o cliente e o servidor se entendam, ou seja, quando o cliente receba uma mensagem do servidor, o mesmo deve saber qual ação deve ser tomada. Para que isso seja possível, são utilizados protocolos nestas mensagens. Protocolos são padrões que devem ser seguidos tanto pelo cliente como pelo servidor para que os dois possam se comunicar de uma forma compatível.

2.4 PROTOCOLOS DE REDE

Quando o conceito de computação distribuída é utilizado, é necessário a escolha de quais protocolos serão utilizados na comunicação entre as máquinas que estão interconectadas, de acordo com Stevens (1993), quando uma mesma linguagem esta sendo utilizada por todo um sistema que possui um comportamento padronizado, um protocolo está sendo utilizado. Como dito anteriormente, a arquitetura a ser utilizada neste trabalho, é a de cliente/servidor, porém ainda é necessário definir quais protocolos serão utilizados nas trocas de mensagens desta rede. Para escolher os protocolos mais adequados a serem utilizados é necessário um entendimento básico de como esses protocolos funcionam assim como suas principais diferenças e vantagens. Devido a essa necessidade, as próximas seções têm como objetivo demonstrar e explicar o funcionamento de alguns dos protocolos mais conhecidos na área de redes.

2.4.1 Protocolo TCP

O protocolo TCP é um protocolo de transporte, orientado a conexões, proporciona uma conexão confiável e envia mensagens de uma forma ordenada (STEVENS, 1993) (TANENBAUM, 2003). Um protocolo orientado a conexão possui a necessidade abrir uma conexão entre as duas máquinas antes de enviar as mensagens, este método torna o uso deste protocolo mais custoso porém com algumas vantagens. Uma dessas vantagens é quanto ao tratamento de erros, o protocolo TCP tem por padrão a verificação de perda de mensagens assim como a ordenação das mesmas.

Caso algum desses erros ocorra, o próprio protocolo se encarrega de corrigilos, ou seja, se a mensagem é perdida durante o percurso entre os computadores a mesma será reenvida até que o outro computador a receba. Isso torna a conexão altamente confiável e com poucas chances de falhas.

Este protocolo é mais utilizado quando há a necessidade de um controle mais intenso sobre as mensagens, como por exemplo, em redes em que a entrega da mensagem é indiscutível. Um exemplo da necessidade deste tipo de rede é uma transação bancária, a perda de uma das mensagens pode acarretar a perda de dados cruciais para o sistema, como o processo de um depósito. Mais informações sobre o protocolo TCP/IP podem ser encontradas em (STALLINGS, 2007).

2.4.2 Protocolo UDP

O protocolo UDP, é um protocolo mais simples em relação ao TCP, ele não tem nenhum tipo de tratamento de erro e também não se preocupa com a sequência das mensagens (STEVENS, 1993). De forma metafórica, o protocolo TCP, é como uma ligação telefônica onde uma conexão é feita entre as duas pessoas e assim que uma delas deixa de responder a outra sabe automaticamente que ocorreu algum tipo de erro com a mensagem enviada. Já o protocolo UDP, é como se as pessoas estivessem trocando cartas, ou seja, não há um controle se a mesma será entregue ou não para seu destinatário.

Essa falta de tratamento torna o protocolo UDP mais simples e rápido, o tornando muito útil para aplicações onde a entrega da mensagem não é tão relevante. Um exemplo é um serviço de chamadas de vídeo como o Skype (2015), as imagens são enviadas via Internet para outro computador, caso uma dessas mensagens se perca no caminho uma pequena fração do vídeo não será mostrada, esta pode ser imperceptível para o olho humano. Devido a isto, neste caso, é melhor ignorar as mensagens perdidas e continuar enviando as novas imagens.

Um outro ponto que diferencia o protocolo UDP do TCP é que o mesmo possui um limite de tamanho para suas mensagens, esse limite é 65,507 *bytes*. Cada uma das suas mensagens são chamadas de datagramas, mais informações sobre o UDP podem ser encontradas em (FOROUZAN, 2002).

2.4.3 Protocolos RTP e RTSP

Esses dois protocolos são um pouco diferentes dos mostrados anteriormente, o primeiro motivo é que eles trabalham em conjunto. Eles são conhecidos como protocolos de controle, onde seu objetivo é controlar as mensagens e não as transferências das mesmas. Devido a isso, uma rede pode utilizar-se dos protocolos UDP, RTP e RTSP em conjunto, onde, o primeiro tomaria conta da parte de envio e recebimento das mensagens e os dois últimos controlariam essas mensagens.

Com o intuito de controlar as mensagens, esses protocolos adicionam novos campos aos pacotes enviados. Mais informações podem ser encontradas em (TANENBAUM, 2003) e (STALLINGS, 2007). Esse novo controle adicionado torna o uso dos outros protocolos melhores dependendo do escopo do problema.

Os protocolos RTP e RTSP são muito utilizados em conjunto com o protocolo UDP devido ao seu enfoque em problemas de *streaming* de vídeo. Alguns dos campos adicionados são para o controle de sequência e de *timestamp*, data e hora de envio e

recebimento. Desta forma, esses protocolos tornam possível o controle sequencial das mensagens mesmo que o protocolo usado seja o UDP. Há outras vantagens e aplicações para os protocolos RTP e RTSP, porém os mesmos não serão discutidos neste trabalho, mais informações sobre eles podem ser encontradas em (SCHULZRINNE, 1998) e (SCHULZRINNE *et al.*, 2003).

2.5 *STREAMING* DE VÍDEO

Wu *et al.* (2001) definem *streaming* como um método de transmitir ou receber dados (especialmente vídeo e áudio) através de uma rede de computadores e que seja possível a execução desses dados enquanto há o recebimento. Esta tecnologia pode ser vista em diversas aplicações na Internet, atualmente alguns exemplos são Spotify, Netflix e Youtube (2015). Eles são serviços que disponibilizam de forma *online* música e vídeo, respectivamente.

Basicamente, *streaming* de vídeo é uma arquitetura cliente/servidor, onde o servidor tem como objetivo enviar *frame* por *frame* de um vídeo para o seus clientes, *frame* é o nome dado para uma única imagem de um vídeo, vários *frames* consecutivos se tornam um vídeo. Apesar da ideia de funcionamento desta tecnologia ser simples a sua implementação requer alguns cuidados que a torna um serviço complexo de ser executado, porém estudos feitos anteriormente facilitam sua implementação devido as várias bibliotecas já existentes para auxiliar na criação de um sistema de *streaming* de áudio/vídeo.

Uma das principais preocupações quando desenvolvemos um servidor de *streaming*, é que o tempo de envio dos *frames* seja baixo, possibilitando uma contínua execução do vídeo para os usuários, esse tempo *frame* necessita desde sua saída do servidor até sua chegada ao cliente é chamado de latência. Uma baixa latência é muito importante para que um *streaming* de vídeo seja eficaz, consequentemente, ao longo deste trabalho maneiras de melhorá-la com o intuito de melhorar o desempenho total do sistema criado.

A velocidade com que o cliente e o servidor trocam informações (latência) tem uma gigantesca importância para o funcionamento do sistema proposto neste trabalho, existem duas maneiras de reduzir o tempo gasto nas trocas de informações entre cliente e servidor: a primeira delas, é o aumento da velocidade da conexão, essa é totalmente dependente da evolução da tecnologia assim como da capacidade financeira do usuário em contratar um serviço de Internet de qualidade. Para que a latência seja baixa, há uma necessidade de que a conexão do usuário seja rápida, tornando o tempo de troca de mensagens rápido o suficiente para que os novos *frames* sejam carregados e mostrados para o usuário de forma imperceptível.

A segunda maneira para reduzir a latência do sistema, é reduzir a quantidade de informações que precisam ser transmitidas entre o cliente e o servidor, essa redução pode ser atingida de diversas maneiras, porém a mais simples delas é a redução do tamanho das mensagem a serem enviadas, utilizando-se de *Codecs* (*Encoder-Decoder*) de vídeo, os *frames* podem ser comprimidos os tornando mais compactos e menores antes de serem enviados pela rede. Essas compressões podem ser feitas de diversas maneiras e algumas delas podem trazer problemas para o sistema, a seção a seguir se encarrega de explicar os conceitos básicos de *codecs* de vídeos.

2.5.1 *Codec* de Vídeo

Como dito na seção anterior, *codecs* de vídeo são utilizados para codificar/decodificar imagens em diversos formatos, a vantagem dessa codificação é a compressão dos dados que representam a imagem.

De forma simplificada, um *codec* de vídeo transforma uma imagem em códigos que quando representados computacionalmente tem um custo menor que o da própria imagem, além de codificar essas imagens os *codecs* também têm como objetivo decodificá-las as tornando legíveis novamente. O processo de codificação pode ou não causar perda na qualidade da imagem, quando o processo de compressão não possibilita uma recuperação total da imagem original (ou seja, perda de qualidade na imagem) ele é chamado (*lossy compression*), e quando o processo de decodificação consegue obter a imagem original a partir de uma imagem codificada o processo de codificação é chamado de (*lossless*).

Vantagens de uma compressão *lossy* é que sua codificação/decodificação geralmente é mais rápida e com maior poder de compactação dos dados, enquanto a compressão *lossless* tem um custo normalmente mais elevado e uma menor capacidade de redução no tamanho da imagem (NELSON; GAILLY, 1996) (SAYOOD, 2012). O uso de cada um desses tipos depende da necessidade do problema, algumas vezes, é melhor uma maior perda de qualidade e uma melhor compressão da imagem e outras vezes é preferível uma melhor qualidade da imagem.

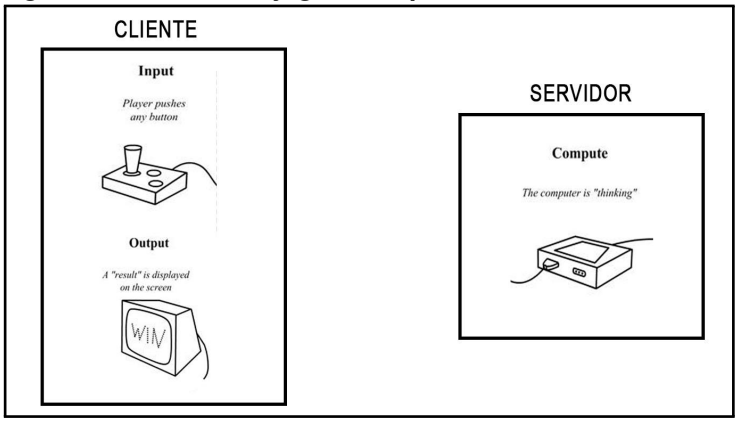
Existem diversos formatos de codificação que são utilizados pelos *codecs* para os vídeos, os mais conhecidos estão listados a seguir: H.265/MPEG-H HEVC, H.264/MPEG-4 AVC, H.263/MPEG-4 Part 2, H.262/MPEG-2 e Google (On2). Cada um desses formatos tem suas vantagens e desvantagens quanto a capacidade de compressão, perda de qualidade da imagem e portabilidade de sistemas. Para mais informações sobre os mesmos consultar (RICHARDSON, 2004).

2.6 JOGOS SOB DEMANDA

Jogos sob demanda ou jogos em nuvens é nome dado a um sistema, onde o jogo é executado utilizando-se do conceito de computação distribuída. O jogo é dividido em duas partes, servidor e cliente (O'BRIEN, 2005).O intuito de um sistema de jogos sob demanda é transferir o custo de processamento do computador do usuário (cliente) para o servidor, isso é possível com o uso de sistemas distribuídos.

Como mostrado anteriormente na Figura 1, o funcionamento de um jogo é dividido em duas etapas, o processamento e o tratamento de entrada e saída. Em um sistema de jogo sob demanda o processamento é totalmente executado pelo servidor, enquanto toda a parte de entrada e saída é tratada pelo cliente. Esta divisão é feita com o intuito de transferir o custo do processamento para um computador mais apto a desenvolver essas tarefas, ou seja, um computador com um poder computacional maior. A Figura 4 mostra como é feita a divisão dos processos entre cliente e servidor.

Figura 4 – Divisão do jogo na arquitetura cliente/servidor.

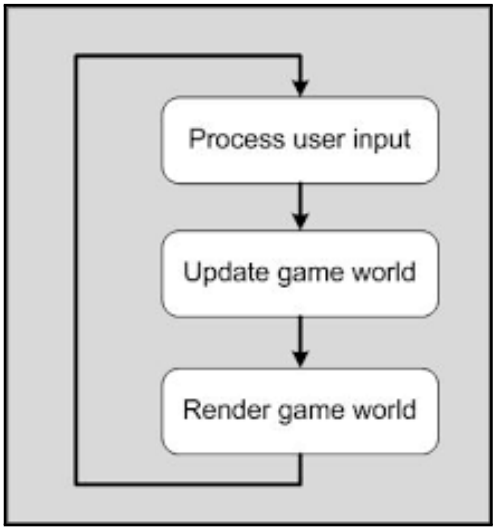


Fonte – Adaptado de Djaouti *et al.* (2008)

O servidor e o cliente se comunicam por meio de mensagens enviadas por uma rede, seja esta a *Internet* ou uma rede local. O servidor tem como objetivo principal o controle do *gameloop*, este é o nome dado para o coração de um jogo. Dentro do *gameloop* é feito todo o processamento do jogo, o mesmo é normalmente dividido em 3 etapas. Essas etapas podem ser vistas na Figura 5.

Todas as etapas do *gameloop* são executadas no servidor, a primeira delas é o processamento da entrada do usuário (*process user input*). A mesma é o processo que decide que ação o personagem dentro do jogo deve tomar, dependendo do botão pressionado pelo jogador. A segunda etapa, é a atualização o mundo virtual (*update game world*), essa é a operação que requer a maior parte do processamento do jogo. Nesta parte, lógicas relacionadas a física e inteligência artificial são aplicadas, essas operações normalmente requerem um grande número de cálculos em um baixo período de tempo. A terceira e última etapa de um *gameloop*, é a renderização *render*

Figura 5 – Ilustração de um *gameloop*.

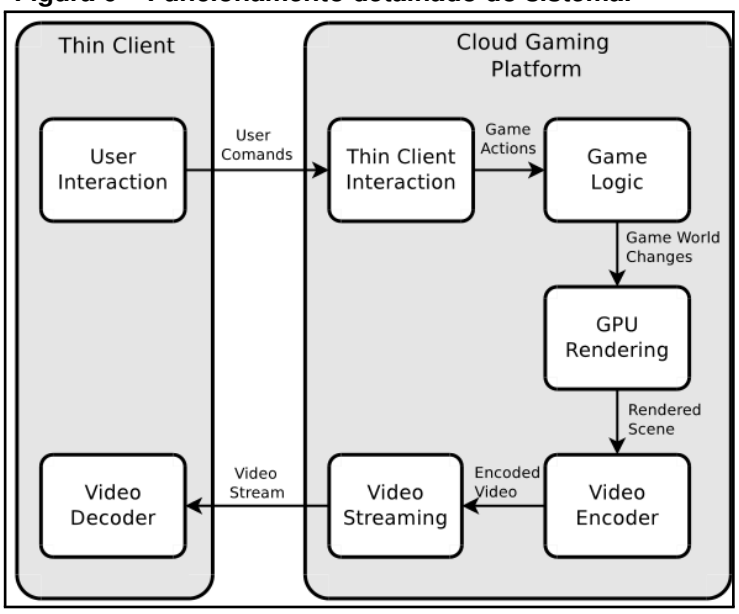


Fonte – Robertson (2016)

game world, este processo é onde os *frames* são preparados e mostrados para o usuário.

No caso de um sistema de jogos sob demanda, o dispositivo que mostra e recebe informações do jogador não é o mesmo computador em que o *gameloop* está sendo executado. Ou seja, depois do processo de renderização dos *frames* os mesmos devem ser enviados para o cliente que deve mostrá-los para o jogador. O mesmo ocorre com as entradas processadas pelo *gameloop*, elas são capturadas pelo dispositivo do cliente que as envia para o servidor por meio da rede para que ocorra a atualização do mundo virtual e a criação dos novos *frames*. A Figura 6 exemplifica o funcionamento mais detalhado deste sistema.

Figura 6 – Funcionamento detalhado do sistema.



Fonte – Shea *et al.* (2013)

O envio dos *frames* do servidor para o cliente acontece por meio de um *streaming* de vídeo, já as entradas do jogador no dispositivo do cliente são enviadas para o servidor por meio de uma conexão via *socket*, utilizando-se de um protocolo pré-definido entre o servidor e o cliente.

3 DESENVOLVIMENTO

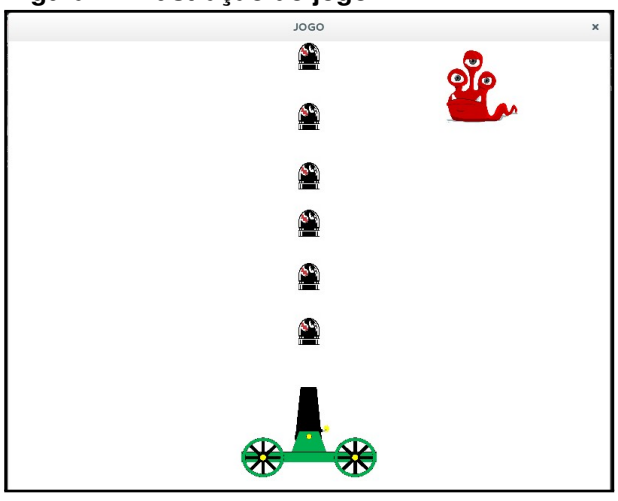
Em todo o desenvolvimento deste trabalho, foi utilizada a linguagem C/C++ juntamente com a biblioteca SDL (*Simple DirectMedia Layer*). A escolha das mesmas se deu porque ambas são *open source* e de baixo nível. Além de existir uma maior afinidade dos desenvolvedores com essas duas ferramentas.

A SDL é uma biblioteca multi-plataforma de baixo nível, criada para auxiliar o acesso a teclado, áudio e placa de vídeo. Em outras palavras é uma biblioteca com o intuito de facilitar o desenvolvimento de jogos, facilitando as interações entre o jogo e os componentes do dispositivo, como teclado e placa de vídeo. Mais informações sobre essa biblioteca podem ser encontradas no site oficial da mesma (SDL, 2016), para o entendimento deste trabalho não é necessário nenhum conhecimento específico da biblioteca SDL.

3.1 O JOGO UTILIZADO

O jogo utilizado para o desenvolvimento deste trabalho, baseia-se em jogos de estilo *arcade*. Ele possui uma única fase onde o jogador utiliza-se de um canhão para atingir um monstro. O mostro fica sempre na parte superior da tela e se movimenta unilateralmente, a cada vez que o jogador consegue atingir o mostro um ponto é concedido para ele. O jogo não possui um final e pode ser jogado infinitamente. A Figura 7 ilustra esse jogo.

Figura 7 – Ilustração do jogo.



Fonte – Autoria própria

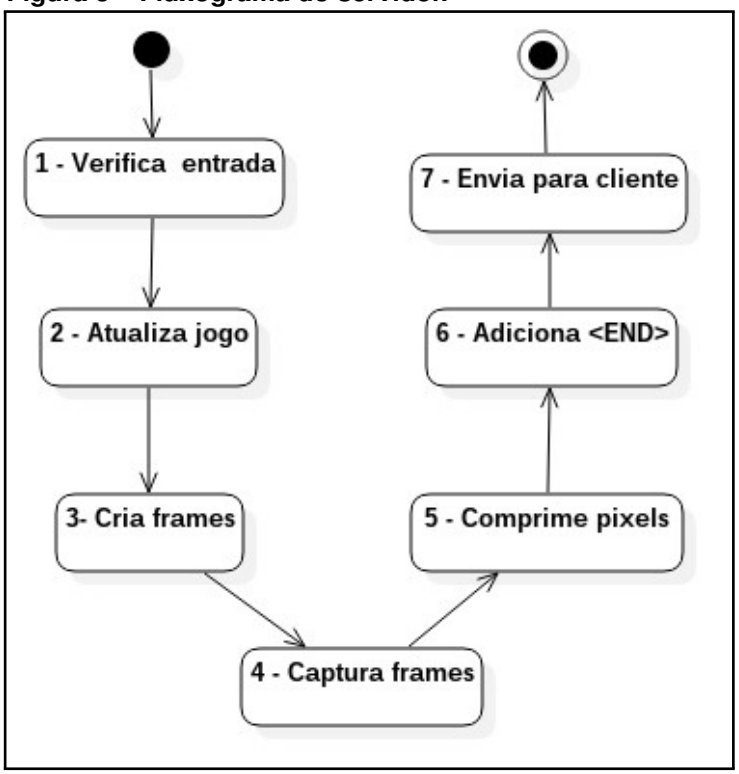
Esse jogo foi escolhido devido a sua simplicidade, tornando o desenvolvimento do sistema mais simples e direto. O mesmo foi desenvolvido utilizando a linguagem C/C++ juntamente com a biblioteca SDL.

3.2 SERVIDOR

O servidor tem como objetivo o processamento do jogo, assim como a captura dos *frames* a serem enviados para o cliente. Neste trabalho foi utilizado uma forma diferente dos métodos mostrados na seção 2.1 para capturar os *frames*: os mesmos são capturados diretamente do *buffer* do jogo, tornando assim o sistema livre da necessidade de criar uma nova máquina virtual para cada nova instância do jogo criada.

Esta forma de captura dos *frames* proporciona vantagens quanto a performance do sistema, porém a mesma necessita de modificações no código fonte de jogo. A Figura 8 mostra o funcionamento geral do servidor.

Figura 8 – Fluxograma do servidor.



Fonte – Autoria própria

A Figura 8 pode ser dividida em duas etapas, a parte da esquerda (quadros 1, 2 e 3), refere-se as lógicas aplicadas ao jogo *gameloop*. A única diferença deste para um jogo normal é que as entradas utilizadas pelo mesmo são recebidas do cliente.

Para ler as entradas o servidor possui uma *thread* que aguarda até que uma ação seja recebida do cliente. Quando essa ação é recebida uma variável é modificada e a ação número 1 do fluxograma se responsabiliza por aplicá-la dentro do jogo. Esse processo é feito dentro de uma *thread*, pois o sistema não pode parar enquanto aguarda as ações do jogador. Mesmo que este não interaja com o jogo, o mesmo tem que atualizar outras informações, como o movimento do monstro na tela.

A cada *gameloop*, novos *frames* são criados, os mesmos são adicionados

em um *buffer* que deve ser mostrado na tela. O servidor também tem como tarefa interceptar os dados desse *buffer* para que o mesmo seja enviado para o cliente, este processo é feito por meio de uma função que a biblioteca *SDL* proporciona para acesso aos *pixels* que estão armazenados no *buffer* do jogo (item 4 no fluxograma).

Após a captura desses *frames* pela função SDL, os mesmos passam por um processo de compressão de dados. A biblioteca *zlibcomplete* foi utilizada no desenvolvimento deste sistema (CILIBRASI, 2016). A mesma implementa o algoritmo *LZ77*. Este é utilizado no processo de compressão de arquivos no formato *.zip* (NELSON; GAILLY, 1996).

Após o processo de compressão, os *frames* são enviados para o cliente, por meio de um *socket*. Esse processo também é feito dentro de uma *thread* para não afetar o *gameloop* que está sendo executado. A cada *frame* enviado para o cliente um delimitador é enviado (”<END>”), o mesmo indica o início e final de cada *frame*. Este delimitador possibilita tratamentos nos dados recebidos pelo cliente, por exemplo para identificar se um *frame* recebido está completo ou não.

3.3 O CLIENTE

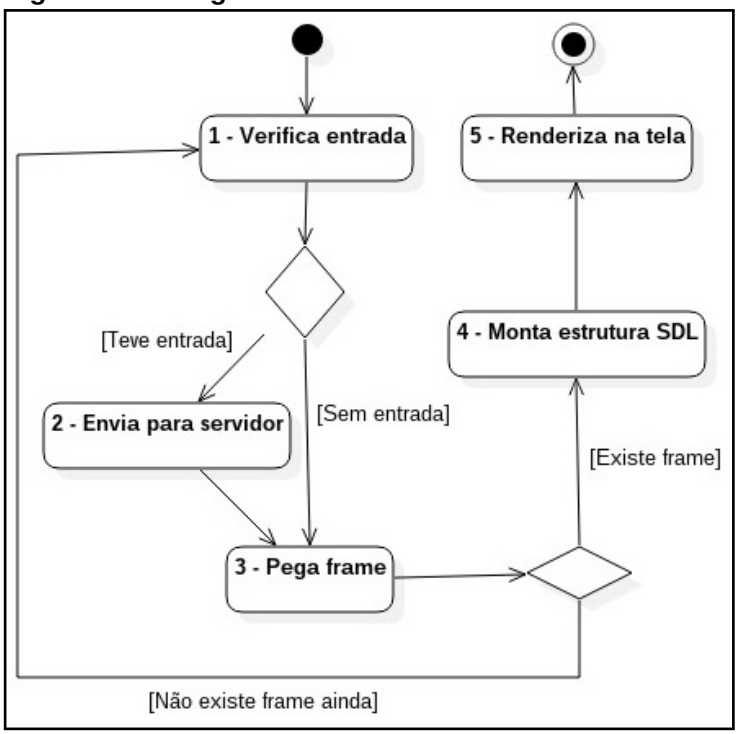
O cliente tem como objetivo interagir diretamente com o usuário, recebendo as ações do mesmo e renderizando os *frames* recebidos pelo servidor na tela. O funcionamento geral do cliente é dividido em dois processos que ocorrem simultaneamente. O primeiro recebe os dados do servidor e monta os *frames* que serão renderizadas pela segunda parte do processo. Além de receber e montar os *frames*, o primeiro processo também se encarrega de enviar as ações executadas pelo jogador para o servidor.

O segundo processo, fica o tempo todo verificando se novos *frames* foram recebidos pelo primeiro processo. Para que os mesmos sejam renderizados na tela o sistema recria a estrutura SDL exatamente igual a que tinha sido criada no servidor. Essa estrutura depende de vários parâmetros que devem ser iguais tanto no servidor quanto no cliente.

Dentre esses parâmetros, estão dados como resolução e qualidade da imagem, que foram configurados manualmente no servidor e no cliente. A dependência de que as duas estruturas SDL, tanto no cliente como no servidor, sejam exatamente iguais foi o principal motivo para a escolha do método de compressão *LZ77*, pois o mesmo é do tipo *lossless*, ou seja, não há perda de dados no processo de compressão, podendo garantir que o cliente vai receber exatamente o mesmo *frame* que o servidor comprimiu. Outros algoritmos de compressão *lossless* como *PNG* não foram utilizados, pois o *LZ77* tem uma melhor taxa de compressão. A Figura 9 mostra o

fluxograma de funcionamento do cliente no sistema.

Figura 9 – Fluxograma do cliente.



Fonte – Autoria própria

Este fluxograma mostra todos os passos do cliente quanto a primeira etapa do processo. A ação número 1 do fluxograma verifica se o jogador executou alguma ação dentro do jogo. Caso a ação tenha acontecido, um código é enviado para o servidor identificando qual ação foi tomada pelo jogador (ação 2), o processo de envio deste código identificador é feito por meio de uma *thread* para não influenciar no processo de atualização dos *frames* do jogo que são feitos pelas ações 3, 4 e 5.

A ação 3 é responsável por verificar se existem *frames* recebidos e prontos para serem renderizados. O processo de receber os *frames* e prepara-los para serem mostrados (descompressão) também acontece dentro de uma *thread* para garantir que o mesmo não influencie na renderização dos *frames*. Caso um novo *frame* não esteja pronto para ser usado quando a ação número 3 é executada, a mesma continua o processo utilizando o último *frame* renderizado.

A recepção dos dados que vão compor o *frame* é feita por meio de uma conexão via *socket*, conforme os dados vão sendo recebidos eles vão sendo adicionados em um *buffer* que deve conter todo o *frame* comprimido. Esse processo ocorre até que o delimitador ”<END>” seja formado pelos dados recebidos.

Após receber o delimitador, os dados do *buffer* são descomprimidos utilizando-se o algoritmo *LZ77*, logo após a descompressão dos dados o *frame* é criado e preparado para ser utilizado dentro do jogo. Neste momento, o mesmo se torna preparado para ser utilizado pela ação número 3 do fluxograma da Figura 9. Após a aquisição

do *frame*, o sistema recria a estrutura *SDL* utilizando as mesmas configurações do servidor e o mesmo é renderizado na tela do usuário (ações 4 e 5).

3.4 COMUNICAÇÃO ENTRE SERVIDOR E CLIENTE

Para o desenvolvimento deste trabalho foi necessário analisar e escolher o melhor protocolo de rede a ser aplicado. Esta seção tem como intuito explicar quais protocolos foram utilizados e o motivo da escolha de cada um deles. Nesta análise, foram levados em consideração os protocolos UDP e TCP, os protocolos RTP e RTSP não foram utilizados, pois os mesmos são uma melhoria do protocolo UDP.

No sistema criado existem dois tipos de envio de mensagens. A primeira delas é originada no cliente. A mesma é utilizada para transferir as ações executadas pelo jogador para o servidor. As mensagens que carregam esses dados são altamente sensíveis, ou seja, podem afetar diretamente na jogabilidade caso perdidas.

Quando o usuário efetua um comando no cliente ele espera que o mesmo seja executado dentro do jogo, como por exemplo se o usuário pressiona o botão de pular seguido do botão para chutar. Essas ações devem acontecer dentro do jogo em que os botões foram pressionados.

Para que as mensagens não sejam perdidas e para que as ações se mantenham em ordem, o protocolo TCP deve ser utilizado para o envio destas mensagens, garantindo assim, a entrega de todas assim como as sequencias em que as mesmas foram enviadas.

Por outro lado, o segundo tipo de envio de mensagem que o sistema utiliza é originado no servidor e não é tão sensível, ou seja, a perda de algumas mensagens não deve afetar diretamente na jogabilidade.

Este envio é referente aos *frames* renderizados no servidor que devem ser enviados e mostrados no cliente, caso aconteça alguma perda de dados neste processo de envio, basta ignorar os dados perdidos e continuar a enviar os novos *frames* gerados.

Graças a alta velocidade com que os *frames* são gerados pelo sistema, a perda de alguns deles é quase que imperceptível. Considerando as necessidades mostradas para o envio dos *frames*, o protocolo UDP deveria ser utilizado neste caso, porém a utilização do mesmo tem alguns pontos negativos em relação ao protocolo TCP.

Devido a limitação dos tamanhos dos datagramas do protocolo UDP, um *frame* teria que ser dividido em partes menores antes de ser enviado. Isso implicaria na necessidade da criação de algoritmos para controlar a sequência do recebimento das

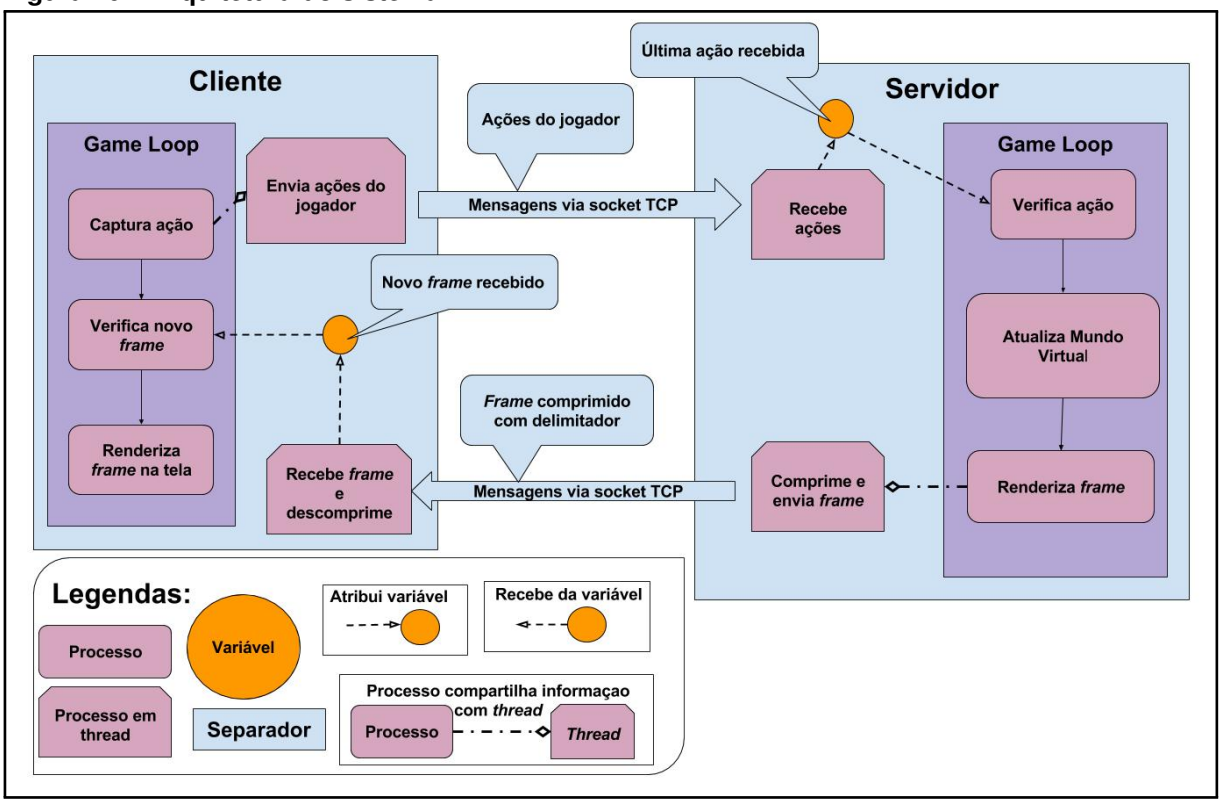
mensagens pelo cliente. A ordem dos dados é importante para o sistema, pois o algoritmo *LZ77* necessita da mesma para que os dados sejam descomprimidos.

Para manter a simplicidade do sistema o protocolo TCP foi escolhido, pois o funcionamento do mesmo não seria tão prejudicial ao sistema. Isso deve-se a grande simplicidade do jogo neste proposto. Porém acredita-se que haveria um pequeno ganho na velocidade do envio dos *frames* utilizando-se do protocolo UDP, assim como uma menor chance de problemas ligados a sincronização e gargalos de rede.

3.5 ARQUITETURA DO SISTEMA

A Figura 10 mostra a visão geral dos processos e *threads* aplicadas no sistema.

Figura 10 – Arquitetura do sistema.



Fonte – Autoria própria

Todas as vezes que um processo compartilha informação com alguma *thread*, uma variável de controle é utilizada com intuito de previnir que os dois processos acessem a informação no mesmo instante.

Todos os processos que enviam e recebem informações são processados dentro de *threads*, isto é feito para não afetar o *game loop*. Tanto o cliente como o servidor continua possuindo o *game loop*, porém pode-se perceber que o *game loop* executado no cliente não executa o processo de atualizar o mundo virtual. O que torna

o processamento do *game loop* no cliente mais rápido, pois atualizar o mundo virtual requer um alto nível computacional.

Alguns problemas foram encontrados na arquitetura criada, os mais relevantes são problemas de sincronização entre cliente e servidor. Dependendo de condições da rede, uma falta de sincronia entre cliente e servidor pode acontecer. Como por exemplo, enquanto o servidor cria o *frame* número 100, o cliente está renderizando o *frame* número 93 para o jogador. O sistema atual não apresenta nenhuma forma de contornar este problema. Uma maneira de resolvê-lo seria ignorar os *frames* de número 94 até o 99, seria o método mais simples porém com perda de bastante *frames*.

Outro problema encontrado, que tem um baixa probabilidade de ocorrer, é o algoritmo de compressão adicionar uma sequência exatamente igual o delimitador (”<END>”) no *buffer*. Caso isso ocorra, o cliente entenderia que um *frame* foi recebido. O sistema por padrão ignora todos *frames* que tenham perdido alguma informação no caminho, essa é uma solução simples para o problema, porém não é a ideal.

4 RESULTADOS

Este capítulo tem como objetivo mostrar os resultados obtidos pelo sistema desenvolvido neste trabalho, o desempenho obtido pelo mesmo foi medido em torno da taxa de FPS por ele obtida nos clientes em diferentes situações, assim como os tempos que influenciaram diretamente no aumento ou diminuição da taxa FPS.

4.1 DISPOSITIVOS UTILIZADOS NOS TESTES

Foram utilizados 3 diferentes dispositivos para os testes, o Quadro 1 mostra cada um deles assim como suas respectivas configurações. O Quadro também disponibiliza um número identificador que será utilizado durante o desenvolvimento deste capítulo para referenciar os dispositivos utilizados nos testes.

Quadro 1 – Dispositivos utilizados para teste

Identificador	Dispositivo	Processador	Memória	Placa de vídeo
1	Lenovo IdeaPad y510	4th Gen Intel® Core™ i7-4700MQ (2.4GHz, Quad Core, Cache 6MB)	8GB DDR3L SDRAM	NVIDIA® GeForce® GT755M 2GB 2x SLI
2	Acer Chromebook 11 C740	Intel® Celeron® 3205U (1.50 GHz, Dual core, Cache 2MB)	4 GB DDR3L SDRAM	Intel® HD Graphics
3	Asus X44C	Intel® Core™ i3 2350M (2.20 GHz, Dual core, Cache 3MB)	4 GB DDR3 SDRAM	Intel® HD Graphics 3000

Fonte – Autoria própria

Os dispositivos 2 e 3 foram utilizados como clientes pois os mesmos possuem menor capacidade computacional. O computador 1 foi utilizado como o servidor do sistema pois o mesmo tem um poder computacional mais elevado comparado com os outros dois dispositivos. Em realidade o dispositivo 1 não pode ser comparado com um servidor real, pois suas configurações não estão nem próximas do necessário. Mas o mesmo pode dar uma base de como o sistema funcionaria em um computador mais potente do que os demais que estão utilizando o sistema, mais informações sobre os dispositivos podem ser encontradas nos detalhes dos produtos nos sites do fabricantes, (LENOVO, 2016), (ACER, 2016) e (ASUS, 2016), respectivamente.

4.2 COMPRESSÃO

Esta seção tem como objetivo mostrar a diferença entre o funcionamento do sistema com e sem compressão. O algoritmo de compressão utilizado foi o *LZ77*. A Tabela 1 mostra o quão eficaz a compressão foi para diferentes resoluções, considerando que a biblioteca SDL foi configurada para utilizar 4 *bytes* para representar cada *pixel*.

Tabela 1 – Comparação de *frames* com e sem compressão.

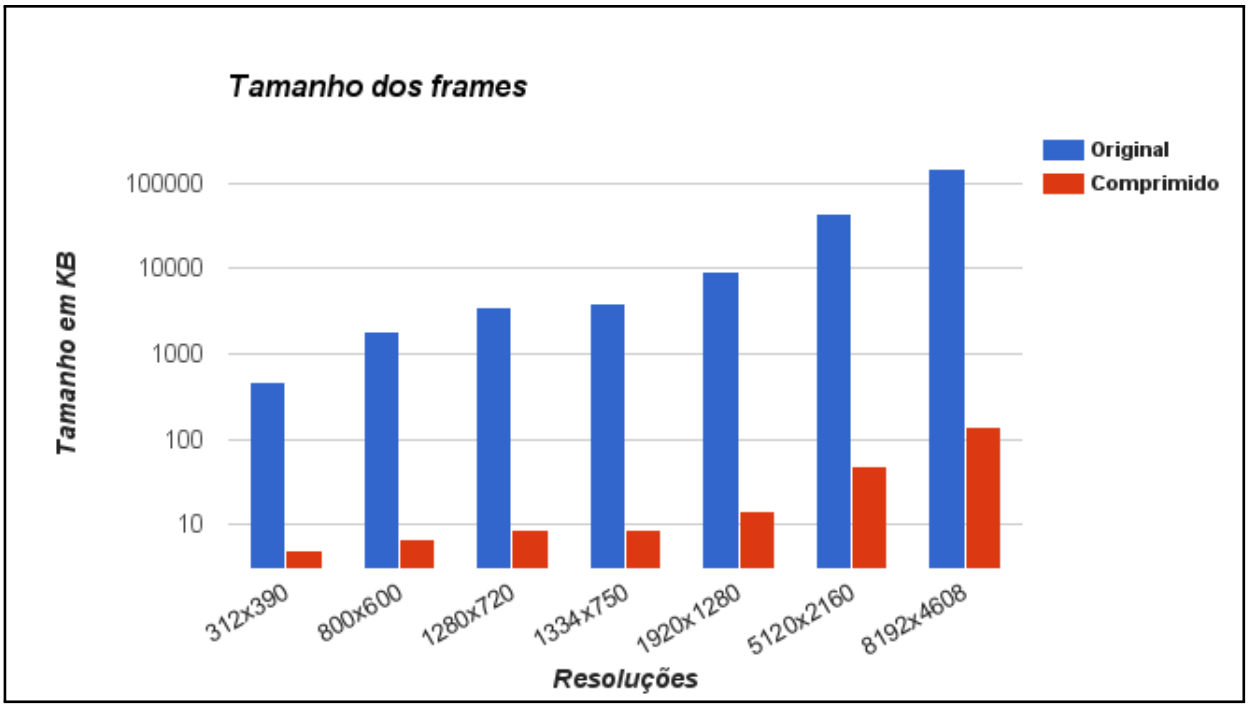
Resolução	Frame original	Frame comprimido
600x400	960000 bytes ou 937,5 KB	5892 bytes ou 5,75 KB
800x600	1920000 bytes ou 1,23 MB	6883 bytes ou 6,72 KB
1024x800	3276800 bytes ou 3,12 Mb	8249 bytes ou 8,05 KB
1920x1200	9216000 bytes ou 8,78 Mb	14072 bytes ou 13,74 KB

Fonte – Autoria própria

Pode-se perceber o quanto a compressão é necessária para o sistema, utilizando uma resolução de 800x600 o *frame* comprimido é 162 vezes menor que o original. É claro que a compressão depende de outras coisas além da resolução, como quantidade de detalhes da imagem assim como número de cores que a mesma possui. Neste sistema o jogo utilizado contém imagens com um baixo número de detalhes e cores, devido a isso a taxa de compressão é tão boa quanto a mostrada. O mesmo dificilmente aconteceria se fosse utilizado um jogo mais sofisticado, porém mesmo que a imagem contenha mais detalhes e cores a taxa de compressão pode chegar a números exorbitantes como 50 vezes menores do que os *frames* originais.

O tamanho dos *frames* está diretamente ligado com a resolução utilizada pelos mesmos, o Gráfico 1 mostra a relação de crescimento do tamanho dos *frames* com e sem compressão em relação a resoluções padrões usadas atualmente.

Gráfico 1 – Comparativo do crescimento do tamanho dos *frames* com e sem compressão.



Fonte – Autoria própria.

Como pode ser observado a velocidade com que o tamanho dos *frames* crescem sem compressão é muito alta, seria praticamente impossível encontrar uma rede

suficientemente rápida para que o sistema funcionasse sem compressão. Ao contrario, a velocidade com que o tamanho dos *frames* comprimidos crescem se mantém instável, não crescendo tão abruptamente.

Com os resultados obtidos no processo de compressão e com o baixo tamanho dos *frames* resultantes deste processo, pode-se ter a impressão que o sistema funcionaria bem com todos os tipos de resoluções.

Porém apesar do tamanho dos *frames* serem suficientemente baixos para serem enviados via rede em uma pequena taxa de tempo, o custo do processamento para compressão e descompressão dos mesmos pode afetar diretamente o funcionamento do sistema.

O processo de compressão e descompressão pode levar um tempo muito grande que ocasione um atraso no envio/criação dos *frames*, a Tabela 2 mostra o tempo que o processo de compressão e descompressão leva dependendo da resolução utilizada. Os dados foram obtidos através de uma média após cinco minutos de funcionamento do sistema, o tempo foi obtido utilizando o tempo de processamento em relação aos ciclos de *clock* do computador. Isso garante que o tempo de processamento não seja influenciado por outros processos sendo executados no dispositivo, o dispositivo número 1 foi utilizado para os testes.

Tabela 2 – Comparativo do tempo médio de compressão e descompressão de um *frame*.

Resolução	Tempo de compressão		Tempo de descompressão		Máximo FPS
	Média	Desvio padrão	Média	Desvio padrão	
600x400	34,87 ms	1,98ms	6,42 ms	0,39ms	25,69
800x600	40,23 ms	2,64ms	7,70 ms	1,08ms	22,61
1920x1280	165,40 ms	4,34ms	49,06 ms	3,98ms	4,85
5120x2160	817,0ms	48,4ms	352,3ms	25,2ms	0.91

Fonte – Autoria própria

Mudanças nas configurações do dispositivo que esta comprimindo ou descomprimindo os *frames* podem influenciar diretamente no tempo de processamento dos mesmos. A Tabela 3 mostra algumas dessas diferenças obtidas com uma resolução padrão de 600x400.

Tabela 3 – Comparativo do tempo médio de compressão e descompressão de um *frame* com resolução 600x400 utilizando diferentes dispositivos.

Dispositivo	Tempo de compressão	Tempo de descompressão
1	34,87 ms	6,42 ms
2	75,15 ms	14,23 ms
3	64,56 ms	12,31ms

Fonte – Autoria própria

Pode-se perceber o quanto o poder computacional influência no processo de compressão, o dispositivo 1 teve um tempo cerca de 2 vezes mais baixo para comprimir e descomprimir os *frames*.

O tempo gasto para a compressão dos dados não é tão relevante para o sistema, pois o mesmo pode ser facilmente reduzido aumentando a capacidade computacional do servidor. Normalmente espera-se que o servidor tenha uma capacidade computacional muito alta, tornando o tempo de compressão dos *frames* realmente baixo.

O grande problema está no tempo de descompressão, pois este processo é feito na máquina do cliente. A mesma normalmente tem um baixo poder computacional como as mostradas no dispositivo 1, o mesmo levou cerca de 558 ms para descomprimir um *frame* em 4K(5120x2160). Desta forma no melhor dos casos uma taxa de 2 de FPS seria alcançada, pois não seria possível descomprimir mais do que 2 *frames* por segundo, tornando inviável a utilização do mesmo.

Utilizando-se de *frames* com resoluções mais baixas como 600x400 o tempo de descompressão cai para 7,64 ms, possibilitando uma taxa de 130 FPS. É claro que a taxa de FPS não depende somente do tempo de descompressão, o mesmo depende de condições da rede entre outras variáveis.

Este tempo de descompressão poderia ser reduzido a zero com o uso de algoritmos de compressão próprios para *streaming* de vídeo, como MPEG e MJPEG. Os mesmos não foram aplicados neste trabalho pela necessidade de recriar a estrutura SDL no cliente, ou seja havia a necessidade de um algoritmo de compressão do tipo *lossless* e não do tipo *lossy* como o Fmpeg e MJpeg.

A utilização de algoritmos de compressão do tipo *lossy* necessita de uma reestruturação do sistema, o que requer um maior tempo de desenvolvimento. Por esse motivo a implementação do mesmo se tornou inviável para este trabalho. O Apêndice A mostra um comparativo entre os métodos de compressão MJpeg e Fmpeg em um sistema de jogos sob demanda, mostrando quais poderiam ser as vantagens utilizando de um desses métodos.

4.3 TEMPO DE ENVIO DOS *FRAMES*

Apesar da capacidade dos computadores processar informação ser um importante ponto para sistemas de jogos sob demanda, a rede ainda é o maior dos problemas. Mesmo com todo o poder de compressão mostrado na seção anterior, o grande fluxo de *frames* torna-se muito custoso para a rede, tornando o tempo de envio uma variável que pode afetar diretamente o funcionamento do sistema.

Neste trabalho foram feitos testes somente em uma rede interna, utilizando-se de uma rede cabeada e uma rede não cabeada. Apesar da distância entre a velocidade de uma rede interna para a Internet, acredita-se que em um futuro próximo

haverá conexão com a Internet tão rápidas quanto as velocidades obtidas hoje em dia nas redes internas.

Foi utilizado o padrão IEE 802.11g para a rede não cabeada utilizada no sistema, o mesmo pode atingir uma velocidade máxima de 52 Mbps podendo ser estendida até 100 Mbps em roteadores mais novos. Com intuito de tentar manter a rede com uma velocidade mais baixa em razão de testes, o padrão antigo sem as melhorias foi utilizado, onde a velocidade máxima atingida pode ser de 52 Mbps (BANERJI; CHOWDHURY, 2013).

A Tabela 4 mostra o tempo médio que *frames* comprimidos com diferentes resoluções necessitam para chegar no destino considerando conexões via Internet, não cabeada (*wireless*) e cabeada com velocidades de 10 Mbps, 54 Mbps e 100 Mbps respectivamente.

Tabela 4 – Média de tempo de envio dos *frames*.

Resolução	Tempo de envio dos frames pela rede		
	Cabeada	Wireless	Internet
600x400	460 μs	851 μs	4,60 ms
800x600	537 μs	995 μs	5,37 ms
1024x800	644 μs	1,19 ms	6,44 ms
1920x1200	1,09 ms	2,03 ms	10,99 ms

Fonte – Autoria própria

Os dados mostrados foram obtidos considerando o tempo de envio dos *frames* comprimidos, Tabela 1, considerando a melhor velocidade possível para cada uma das redes. Essa métrica pode proporcionar de forma superficial os tempos necessários para o envio dos *frames* pela rede, porém este tempo pode ser influenciado por inúmeros fatores, como por exemplo o número de pessoas utilizando a rede em determinado momento.

Como pode ser visto, os tempos de envio se mantiveram baixos para a maioria das redes, isso pode ser obtido pois a taxa de compressão dos *frames* neste sistema é bem alta.

Mesmo com uma alta taxa de compressão podemos perceber que houve uma mudança significativa do tempo de envio utilizando a conexão com Internet, o *frame* com resolução 1920x1200 levou cerca de 12 vezes o tempo do *frame* com resolução 600x400. Como o tamanho dos *frames* nesse sistema são baixos, devido a alta taxa de compressão, a diferença de tempo se mantém dentro de um limite que possibilita o funcionamento do sistema. Porém mesmo não aconteceria com *frames* mais ricos em detalhes.

Analisando exclusivamente o tempo de envio dos *frames* obtidos com o jogo utilizado neste trabalho seria possível obter taxas bem altas de FPS, porém como visto na seção anterior o tempo gasto para envio dos *frames* não é o único empecilho

no sistema criado. O tempo de descompressão no caso deste trabalho se tornou o limitador da taxa de FPS. A próxima seção mostra os resultados obtidos pelo sistema utilizando a taxa de FPS como parâmetro.

4.4 TAXA DE FPS

Quando se fala de jogos preocupa-se muito com a taxa de FPS, o mesmo mostra o quão fluído o jogo é para o usuário. Uma taxa de FPS entre 30 e 60 é considerada imperceptível para o olho humano, o Quadro 2 mostra a variação das taxas de FPS obtidas pelo sistema (n-m), onde n foi a pior taxa obtida e m a melhor delas. O mesmo foi testado em diferentes ambientes e utilizando-se de diversas resoluções, nota-se que foi utilizado o simbolo X para indicar quando um dispositivo não possuía suporte para determinado tipo de conexão.

Quadro 2 – Variação nas taxas de FPS obtidas nos testes.

Resolução	Dispositivo	Taxa de FPS na rede	
		Wireless	Cabeada
600x400	2	25-35	X
	3	26-38	32-40
800x600	2	16-22	X
	3	19-26	20-28
1024x800	2	11-13	X
	3	12-15	13-17

Fonte – Autoria própria

Em alguns casos o sistema obteve uma taxa maior que 30 FPS, mostrando que o sistema pode ser usado em determinadas condições. Porém como pode-se perceber que com o aumento das resoluções o qualidade do sistema deixa a desejar, esse problema já foi mostrado na seção 4.2 e uma possível solução para o mesmo é aplicar outro algoritmos de compressão de dados.

Outro dado importante foi a leve melhora obtida pela rede cabeada, normalmente quando ela é utilizada as taxas de FPS são melhores. Entretanto nos testes sobre o sistema desenvolvido neste trabalho o tempo de descompressão foi o limitante do crescimento da taxa de FPS, o mesmo ocorre pelas condições de rede utilizadas nos testes assim como pela simplicidade do jogo.

Os dados obtidos neste trabalho, mostram que apesar dos inúmeros problemas que o sistema enfrenta é possível o funcionamento do mesmo. Ainda há uma falta de tecnologias em redes e processamento para que o mesmo se torne excelente, mas os resultados aqui mostrados provam que em um futuro próximo esses sistemas estarão fazendo parte do mundo dos jogos trazendo mais facilidades tanto para os desenvolvedores como para os jogadores.

5 CONCLUSÃO

Os serviços nos dias de hoje estão passando por uma fase de mudanças. As pessoas cada vez mais preferem acessa-los via *web* do que baixar aplicativos em seus computadores e/ou *smartphones*. Uma grande parte desses serviços, já são disponibilizados via Internet. Exemplos desses são serviços de vídeo e áudio, como Netflix e Spotify. Porém, ainda, há uma falta de serviços similares em outras áreas, como as de jogos digitais. O principal motivo para essa falta é a dificuldade que é encontrada para desenvolver sistemas como o proposto neste trabalho. Esses tipos de sistemas normalmente dependem de inúmeras variáveis o que dificulta a criação/manutenção do mesmo.

Problemas encontrados neste sistema, como sincronia entre cliente e servidor, compressão de dados e forma de envio/captação dos *frames*, são complexos de serem resolvidos e requerem um conhecimento amplo de várias áreas da computação, desde processamento de imagens até a área de redes.

Levando em consideração os resultados neste trabalho obtidos, pode-se perceber que sistemas de jogos sob demanda já são realidade nos dias de hoje. Taxas de 35 *frames* por segundo são consideradas quase que imperceptíveis aos olhos humanos, podendo proporcionar uma jogabilidade adequada para os usuários.

O grande limitante do crescimento da taxa de *frames* neste trabalho foi o tempo para a compressão dos *frames*, o mesmo pode ser melhorado utilizando-se de *hardwares* mais potentes, ou com a utilização de outras formas de compressão. Considerando os dispositivos utilizados neste trabalho, pode-se concluir que o sistema seria ainda melhor em configurações mais realísticas.

5.1 TRABALHOS FUTUROS

Esta seção tem como objetivo mostrar trabalhos futuros que poderiam ser desenvolvidos a partir do sistema criado neste trabalho.

- Como mostrado nos resultados do sistema, o processo de descompressão pelo cliente pode ser muito custoso para o computador/dispositivo. Esse processo poderia ser mais rápido utilizando-se de algoritmos de compressão do tipo *lossy*, como por exemplo MJPEG ou MPEG. Para que isso fosse possível, haveria a necessidade de modificar a maneira com que os *frames* são renderizados pelo cliente, onde a estrutura SDL fosse configurada automaticamente dependendo dos *frames* recebidos.

- O sistema criado neste trabalho não pode ser configurado pelo cliente. Configurações como resolução da tela e qualidade da imagem devem ser configuradas manualmente. Poderia-se criar um sistema de controle que cuidasse dessas configurações e que além disso executasse uma nova instância do jogo para cada cliente conectado. Atualmente o sistema depende da iniciação manual do jogo no servidor.

- O sistema criado tem uma dependência da biblioteca SDL. Outras bibliotecas como OpenGL poderiam ser utilizadas, tornando o sistema mais adepto para um número maior de jogos. Esta mudança seria grande, pois haveria a necessidade de se criar uma nova maneira de adquirir os dados do *buffer* do jogo, porém grande parte do sistema poderia ser reaproveitado.

- Neste trabalho, o cliente foi idealizado/criado para as plataformas *desktop* (Windows e Linux), porém nada impediria o sistema funcionar em outras plataformas, tais como, a Internet e os *smartphones*.

REFERÊNCIAS

ACER. **Acer Chromebook 11 C740**. 2016. Disponível em: <http://www.acer.com/ac/en/US/content/model/NX.EF2AA.002>. Acesso em: 3 Out. 2016.

ARMBRUST, M. *et al.* Above the clouds: A berkeley view of cloud computing. **Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS**, v. 28, p. 13, 2009.

ASUS. **Asus X44C - Specifications**. 2016. Disponível em: <https://www.asus.com/Notebooks/X44C/specifications/>. Acesso em: 3 Out. 2016.

BANERJI, S.; CHOWDHURY, R. S. On ieee 802.11: Wireless lan technologys. **International Journal of Mobile Network Communications and Telematics**, v. 3, n. 4, 2013.

BARBOZA, D. C. *et al.* A simple architecture for digital games on demand using low performance resources under a cloud computing paradigm. In: IEEE. **Games and Digital Entertainment (SBGAMES), 2010 Brazilian Symposium on**. [S.l.], 2010. p. 33–39.

BERSON, A. **Client/Server Architecture**. New York, NY, USA: McGraw-Hill, Inc., 1992. ISBN 0-07-005076-7.

CILIBRASI, G. rudi. **C++ interface to the ZLib library**. 2016. Disponível em: <https://github.com/rudi-cilibrasi/zlibcomplete>. Acesso em: 10 Fev. 2016.

CLAYPOOL, M.; CLAYPOOL, K.; DAMAA, F. The effects of frame rate and resolution on users playing first person shooter games. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. **Electronic Imaging 2006**. [S.l.], 2006. p. 607101–607101.

COMMUNICATIONS, A. **An explanation of video compression techniques**. 2008. Disponível em: <http://www.axis.com/files/whitepaper/wp_videocompression_33085_en_0809_lo.pdf>. Acesso em: 22 Jun. 2015.

COULOURIS, G. *et al.* **Sistemas Distribuídos: Conceitos e Projeto**. [S.l.]: Bookman Editora, 2013. v. 5.

DJAOUTI, D. *et al.* A gameplay definition through videogame classification. 2008.

EXAME. **Dados em dispositivos móveis devem crescer 300% até 2017**. 2015. Disponível em: <http://exame.abril.com.br/tecnologia/noticias/dados-em-dispositivos-moveis-devem-crescer-300-ate-2017>. Acesso em: 20 Out. 2015.

FOROUZAN, B. A. **TCP/IP protocol suite**. [S.l.]: McGraw-Hill, Inc., 2002.

FREEK, C. *et al.* On the accuracy of a mjpeg-based digital image compression piv-system. **Experiments in Fluids**, Springer, v. 27, n. 4, p. 310–320, 1999.

GALL, D. L. Mpeg: A video compression standard for multimedia applications. **Communications of the ACM**, ACM, v. 34, n. 4, p. 46–58, 1991.

GOOGLE. **Google**. 2015. Disponível em: <https://www.google.com/>. Acesso em: 20 Out. 2015.

HUANG, C.-Y. *et al.* Gaminganywhere: An open cloud gaming system. **ACM Multimedia Systems**, 2013.

LENOVO. **Lenovo Y510p Laptop - Tech Specs**. 2016. Disponível em: <http://shop.lenovo.com/us/en/laptops/lenovo/y-series/y510p/#tab-tech_specs>. Acesso em: 3 Out. 2016.

NELSON, M.; GAILLY, J.-L. **The data compression book**. [S.l.]: M&T Books New York, 1996. v. 2.

NETFLIX. **Netflix**. 2015. Disponível em: <https://www.netflix.com/>. Acesso em: 20 Out. 2015.

NEWTON, S. **A beginners guide to frame rates**. 2013. Disponível em: <http://aframe.com/blog/2013/07/a-beginners-guide-to-frame-rates/>. Acesso em: 20 Out. 2015.

NICK, S. **Sony buys OnLive streaming game service, which will shut down later this month**. 2015. Disponível em: <http://www.cnet.com/news/sony-buys-onlive-streaming-game-service-which-will-shut-down-later-this-month/>. Acesso em: 25 Out. 2015.

NVIDIA. **Geforce Now: The New Way to Game**. 2015. Disponível em: <http://shield.nvidia.com/game-streaming-with-geforce-now>. Acesso em: 20 Set. 2015.

____. **True Virtual Acceleration With GPUs**. 2015. Disponível em: <http://www.nvidia.com/object/grid-technology.html>. Acesso em: 13 Set. 2015.

O'BRIEN, R. **Gaming on demand system and methodology**. [S.l.]: Google Patents, 2005. US Patent App. 11/145,845.

RICHARDSON, I. E. **H. 264 and MPEG-4 video compression: video coding for next-generation multimedia**. [S.l.]: John Wiley & Sons, 2004.

ROBERTSON, J. **Pygame stuff**. 2016. Disponível em: <https://retrosnob.wordpress.com/2014/11/07/pygame-stuff/>. Acesso em: 17 Out. 2016.

SAYOOD, K. **Introduction to data compression**. [S.l.]: Newnes, 2012.

SCHULZRINNE, H. Real time streaming protocol (rtsp). 1998.

SCHULZRINNE, H. *et al.* **RTP: A transport protocol for real-time applications**. [S.l.], 2003.

SDL. **About SDL**. 2016. Acessado em 10 de Setembro. Disponível em: <https://www.libsdl.org/>. Acesso em: 10 Sep. 2016.

SHEA, R. *et al.* Cloud gaming: architecture and performance. **IEEE Network**, IEEE, v. 27, n. 4, p. 16–21, 2013.

SKYPE. **Skype**. 2015. Disponível em: <http://www.skype.com/>. Acesso em: 20 Out. 2015.

SONY. **Playstation Now**. 2015. Disponível em: <https://www.playstation.com/en-us/explore/playstationnow/>. Acesso em: 20 Out. 2015.

SORIANO, R. **Conhecam o popcorn time e porque o netflix teme sua popularização**. 2015. Disponível em: <http://renansoriano.com.br/blog/conhecam-o-popcorn-time-e-porque-o-netflix-teme-sua-popularizacao/>. Acesso em: 20 Out. 2015.

SPOTIFY. **Spotify**. 2015. Disponível em: <https://www.spotify.com/>. Acesso em: 20 Out. 2015.

STALLINGS, W. **Data and computer communications**. [S.l.]: Pearson/Prentice Hall, 2007.

STATISTA. **Number of worldwide internet users from 2000 to 2015 (in millions)**. 2015. Acessado em 19 de Setembro. Disponível em: <http://www.statista.com/statistics/273018/number-of-internet-users-worldwide/>. Acesso em: 18 Out. 2015.

____. **Video games revenue worldwide from 2012 to 2015, by source (in billion U.S. dollars)**. 2015. Disponível em: <http://www.statista.com/statistics/278181/video-games-revenue-worldwide-from-2012-to-2015-by-source/>. Acesso em: 18 Out. 2015.

STEVENS, W. R. **TCP/IP Illustrated (Vol. 1): The Protocols**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1993. ISBN 0-201-63346-9.

TANENBAUM, A. S. **Computer networks, 4-th edition**. Amsterdam: Prentice-Hall, 2003.

TANENBAUM, A. S.; STEEN, M. V. **Distributed systems: Principles and Paradigms**. Amsterdam: Prentice-Hall, 2007.

TAURION, C. **Cloud Computing: computação em nuvem: transformando o mundo da tecnologia da informação**. [S.l.]: Brasport, 2009.

WALLACE, G. K. The jpeg still picture compression standard. **IEEE transactions on consumer electronics**, IEEE, v. 38, n. 1, p. xviii–xxxiv, 1992.

WU, D. *et al.* Streaming video over the internet: approaches and directions. **IEEE Transactions on circuits and systems for video technology**, IEEE, v. 11, n. 3, p. 282–300, 2001.

YOUTUBE. **Youtube**. 2015. Disponível em: <https://www.youtube.com/>. Acesso em: 20 Out. 2015.

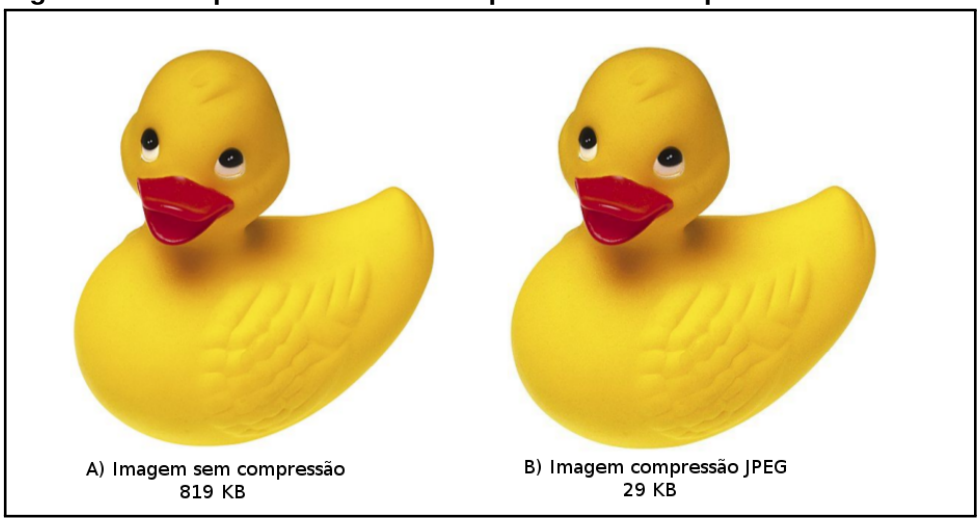
APÊNDICE A - Comparativo entre MJPEG e MPEG

A.1 MJPEG

MJPEG é um dos algoritmos de compressão de vídeos mais conhecidos nos dias de hoje, seu nome é uma referência a *Motion JPEG*. JPEG, é um algoritmo de compressão de imagens mais utilizado atualmente, o seu formato é dado pela extensão *.jpg*, o mesmo é amplamente utilizado na Internet pelo seu grande poder de compressão com uma baixa perda de qualidade.

A sua alta taxa de compressão, é devida ao método de codificação *lossy*, porém a perda de qualidade são praticamente imperceptíveis ao olho humano. Neste trabalho, não haverá uma explicação aprofundada do funcionamento dos algoritmos de compressão, pois o foco do trabalho são os resultados finais de cada um dos métodos. A Figura 11 mostra a comparação de uma imagem sem compressão e outra utilizando-se da compressão JPEG.

Figura 11 – Comparativo de tamanho/qualidade da compressão JPEG.



Fonte – Autoria própria

Como pode ser visto no Figura 11, apesar da perda de qualidade proporcionada pelo método de compressão *lossy*, o mesmo é quase que imperceptível, pode-se notar apenas uma pequena perda na luminosidade da imagem e minúsculos ruídos adicionados a mesma. Porém pode-se notar que a diferença de tamanho dos arquivos é extremamente alta, tornando a perda de qualidade viável para o alto nível de compressão.

O MJPEG é um método que cria uma *stream* de imagens JPEG, de forma mais simples, este método coloca várias imagens JPEG para serem mostradas em sequência. Sendo assim, um vídeo compactado pelo algoritmo MJPEG pode ser descrito como várias imagens JPEG em sequencia, como cada uma dessas imagens passarão por um processo de compressão, o JPEG, o vídeo em geral também fica compactado.

Uma das principais vantagens desse método é a sua simplicidade, pois cada *frame* do vídeo é uma imagem no formato *.jpg*. Um ponto importante é que cada um desses *frames* são independentes entre si, o que diferencia esse método de outros. Mais detalhes sobre o funcionamento do MJPEG e do algoritmo de compressão do JPEG podem ser encontrados em (FREEK *et al.*, 1999) e (WALLACE, 1992) respectivamente.

A.2 MPEG

MPEG ou *Moving Picture Experts Group*, é um grupo voltado para criar padrões para a compressão de vídeos, o mesmo foram os criadores do algoritmo de compressão MPEG 4. O mesmo tem *.mp4* como seu formato, é o formato de vídeo mais conhecido pelos usuários de todo o mundo. Diferente do método de compressão do MJPEG, o MPEG se baseia em uma junção mais complexas dos *frames* com o intuito de se criar o vídeo final. Isso significa, que este algoritmo não simplesmente coloca imagens comprimidas em sequência como é feito no MJPEG, ele verifica repetições entre as imagens não precisando assim replicar a mesma imagem mais de uma vez. A Figura 12 mostra um exemplo deste funcionamento.

Figura 12 – Funcionamento da compressão MPEG.

Fonte – Communications (2008)

Como pode-se notar, o algoritmo leva em consideração que o único objeto que está mudando na cena é o rapaz correndo, então o mesmo guarda somente ele nos próximos *frames*, pois o restante da imagem já foi guardada em algum *frame* anterior. Tornando assim o algoritmo com um poder de compressão ainda maior, pois além de comprimir cada uma das imagens guardadas ele também reduz a repetição das mesmas. A forma com que esses *frames* são escolhidos não será explicada aqui pois seu funcionamento é complexo e esse não é o objetivo do trabalho, porém mais informações podem ser encontradas em (RICHARDSON, 2004) e (GALL, 1991).

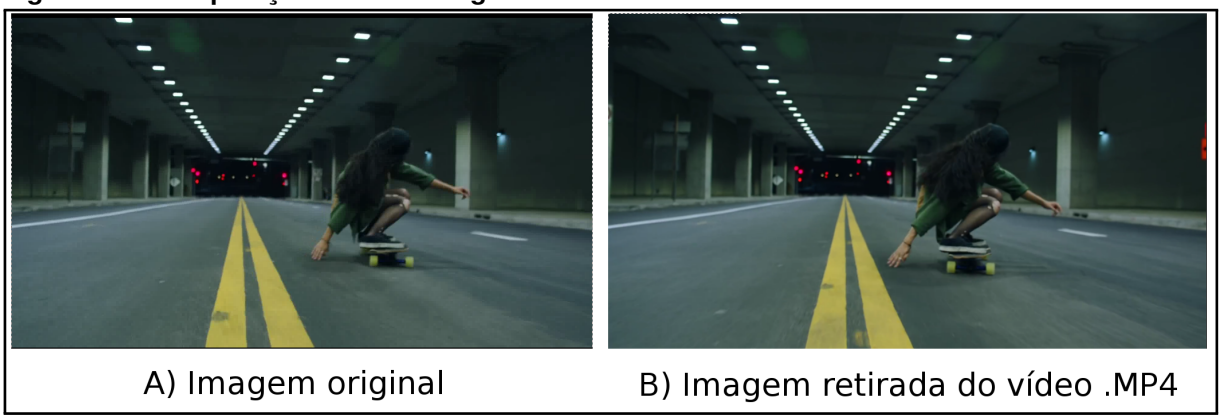
Para que o MPEG possa realizar as comparações entre os *frames* e decidir

quais não se repetem há uma necessidade de obter alguns *frames* antes de começar o processo de compressão dos mesmos, este normalmente não é um problema quando o sistema é utilizado simplesmente para armazenamento de arquivos em disco ou quando ele é utilizado para um *streaming* de vídeo como o do *Youtube*, pois o usuário pode esperar alguns segundos até que alguns *frames* tenham sido gerados antes da transmissão começar. Porém para um sistema de jogos sob demanda essa necessidade de espera para obter alguns *frames* pode causar algumas falhas de sincronização entre o servidor e o cliente, e utilizar um número menor de *frames* para serem comparados pode afetar a capacidade de compressão do método.

A.3 RESULTADOS

Esta seção tem como intuito mostrar os resultados obtidos na comparação dos dois métodos de compressão, os atributos que foram utilizados para os testes foram qualidade das imagens e tamanho total do arquivo após a compressão em cada um dos algoritmos. Como já mostrado na Figura 11, as imagens comprimidas pelo processo do MJPEG perdem pouca qualidade, tornando este um bom algoritmo para compressão de vídeos levando em consideração qualidade final das imagens. Porém o algoritmo de compressão utilizado pelo MPEG não fica muito atrás neste requisito, a Figura 13 mostra a comparação entre uma imagem do vídeo original e outra de um vídeo no formato *.MP4*.

Figura 13 – Comparação do vídeo original e no formato *.MP4*

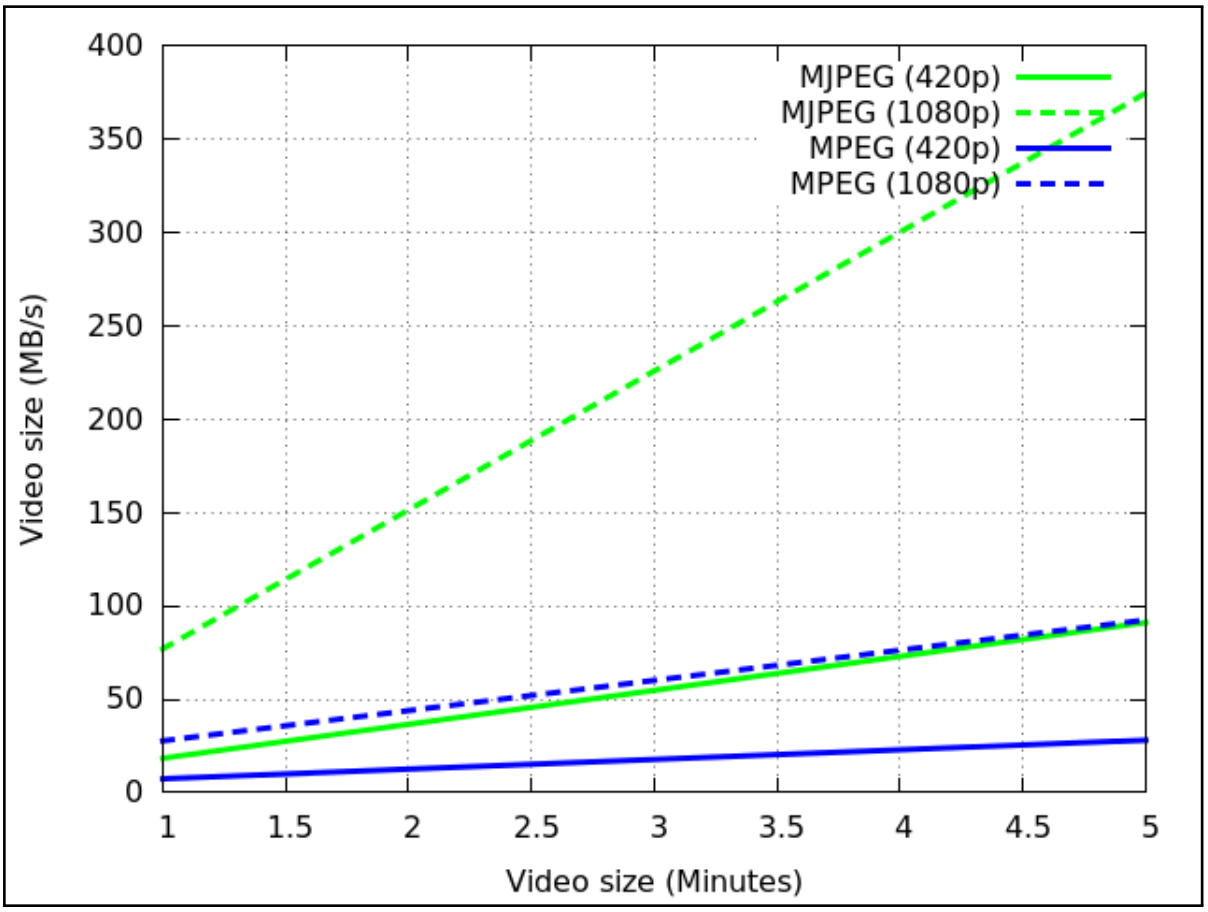


Fonte – Autoria própria

Como pode ser visto na Figura 13 é quase que imperceptível alguma diferença nas duas imagens, tornando a qualidade final das imagens bem similares nos dois métodos. Então para ter uma melhor avaliação dos dois métodos foi feito um estudo levando em consideração o tamanho final de cada arquivo, para isso dois vídeos foram utilizados. O primeiro contendo 1 minuto e o segundo contendo 5 minutos, ambos os vídeos não possuem áudio pois o mesmo poderia desbalancear as medidas finais.

Cada um dos vídeos foi analisado utilizando duas qualidades de imagem (*420p* e *1080p*), ambas qualidades são abrangentemente utilizadas nos dias de hoje podendo ser encontradas em serviços como *Youtube* e *Netflix*. O Gráfico 2 compara o tamanho final de cada um dos arquivos utilizados nos testes.

Gráfico 2 – Comparando tamanho nos formatos *.MP4* e *.MJPEG*



Fonte – Autoria própria.

Como o gráfico mostra, o desempenho do método MPEG se dá muito melhor que o MJPEG quando levado em consideração um vídeo com uma alta quantidade de detalhes como os vídeos em 1080p, esta vantagem se dá devido a utilização do conceito de comparação entre os *frames* que o MPEG aplica. Pode-se perceber que quando trata-se de vídeos curtos (1 minuto) ou vídeos com poucos detalhes como os de 420p, apesar do MPEG continuar na frente a diferença entre os dois é bem menor. Isto acontece devido a menor quantidade de *frames* que o MPEG deixou de criar devido as comparações de padrões entre os *frames*.

Neste trabalho não foi testado como os dois métodos se adaptavam com vídeos que não seguiam um padrão entre os *frames*, porém acredita-se que neste caso o método MJPEG conseguiria diminuir ainda mais a diferença entre os tamanhos dos arquivos devido ao baixo número de padrões que vão ser encontrados no vídeo.

Apesar de o método MPEG ser muito mais eficiente quanto a compressão

dos dados ele requer um tempo de codificação um pouco maior que o do MJPEG, isso se da devido ao seu processo de comparação e escolha entre *frames*. Nos testes realizados neste trabalho a codificação dos arquivos no formato *.MJPEG* levou cerca de $3/4$ do tempo levado pela codificação dos arquivos em *.MP4*.

Levando em consideração os resultados percebe-se que o MPEG tem uma qualidade de imagem tão boa quanto o do método MJPEG e possui um poder de compactação, porém o MPEG enfrentaria um problema ao ser utilizado em um sistema de jogos sob demanda, este é devido a baixa quantidade de *frames* que o mesmo teria para executar suas comparações. Isso se acontece pois o sistema de jogos sob demanda cria e envia os *frames* em tempo real, o que faria com que os resultados obtidos pelo MPEG fossem menores.

Existem formas de contornar esse problema como a criação de um *buffer* de imagens antes de enviá-las para o cliente, porém isso faria com que a complexidade do código do sistema aumentasse pois seria necessário adicionar lógicas para tratar problemas de sincronia entre cliente e servidor.

Um ponto que pode ser chave no momento de escolher um dos dois métodos para um sistema de jogos sob demanda é os jogos que o mesmo pretende utilizar, pois em jogos que são divididos em turnos como xadrez o MPEG se daria muito bem. Isso ocorre pois jogos de turnos não necessitam de uma alta sincronia em cliente e servidor, sendo assim o sistema teria um tempo para criar uma sequência de imagens antes de enviá-las para o cliente. Porém se este sistema tem como foco jogos de FPS (*First Person Shooter*), onde é necessário uma alta sincronia entre cliente e servidor o método MJPEG se daria melhor.