Sistemas de Multi-projeção imersivos e interativos baseados em tecnologias web

José Hamilton Mota

Maio / 2017

Dissertação de Mestrado em Ciência da Computação

Sistemas de Multi-projeção imersivos e interativos baseados em tecnologias *web*

Esse documento corresponde à dissertação apresentada à Banca Examinadora no curso de Mestrado em Ciência da Computação da Faculdade Campo Limpo Paulista.

Campo Limpo Paulista, 26 de maio de 2017.

José Hamilton de Souza Mota Junior

Prof. Dr. Marcelo de Paiva Guimarães
(Orientador)

FICHA CATALOGRÁFICA

Dados Internacionais de Catalogação na Publicação (CIP)

Câmara Brasileira do Livro, São Paulo, Brasil.

Mota Jr., José Hamilton de Souza

Sistemas de multi-projeção imersivos e interativos baseados em tecnologias web / José Hamilton de Souza Mota Jr. Campo Limpo Paulista, SP: FACCAMP, 2017.

Orientador: Profº. Dr. Marcelo de Paiva Guimarães

Dissertação (Programa de Mestrado em Ciência da Computação) – Faculdade Campo Limpo Paulista – FACCAMP.

1. Realidade Virtual. 2. Cluster de computadores. 3. Aglomerados gráficos. 4. CAVE. 5. Multi-projeção. 6. Web. 7. HTML5. 8. JavaScript. 9. WebGL. 10. WebRTC. 11. Adversidades de redes. I. Guimarães, Marcelo de Paiva. II. Campo Limpo Paulista. III. Título.

CDD-005.2

Resumo: *As aplicações de Realidade Virtual imersivas e interativas são baseadas em clusters de computadores e executadas em ambientes de multi-projeção, tais como CAVE (Cave Automatic Virtual Environment) e Video wall. Este trabalho visa explorar a possibilidade de utilizar de forma eficaz tecnologias web para o desenvolvimento de aplicações de multi-projeção (HTML5, JavaScript, WebGL e WebRTC). Para isso, foi idealizada uma arquitetura que visa auxiliar o desenvolvimento destas aplicações visando a sua execução em navegadores web, que proporciona de forma simplificada os recursos de hardware e rede que a aplicação precisar para ser executada corretamente. Também foram avaliados os requisitos de comunicação e sincronização dessas aplicações que são tradicionalmente o principal gargalo, através de testes de desempenho de rede como perda de pacotes, largura de banda e latência. Como prova de conceito para mostrar a viabilidade deste estudo, foi projetada e implementada uma aplicação de realidade virtual imersiva e interativa utilizando apenas tecnologias web, executada em um cluster e projeta em um mini-CAVE com três telas.*

Palavras-chave*: Realidade Virtual, Cluster de Computadores, Aglomerados Gráficos, CAVE, Multi-projeção, Web, HTML5, JavaScript, WebGL, WebRTC, Adversidades de Redes.*

Abstract: *The Immersive and interactive Virtual Reality applications are based on computer clusters and run in multi-projection environments such as Cave Automatic Virtual Environment (CAVE) and Video wall. This work aims to explore the possibility of effectively using web technologies for the development of multi-projection applications (HTML5, JavaScript, WebGL and WebRTC). For this, an architecture was designed to help the development of these applications for execution in web browsers, which provides in a simplified way the hardware and network resources that the application needs to be executed correctly. We also evaluated the communication and synchronization requirements of these applications, which are traditionally the main bottleneck, through network performance tests such as packet loss, bandwidth and latency. As a proof-of-concept to show the feasibility of our study, we devised and implemented an immersive and interactive virtual reality application employing only web technologies and run on a mini-CAVE environment with three displays.*

Keywords*: Virtual Reality, Cluster of Computers, Graphics Agglomerates, CAVE, Multi-projection, Web, HTML5, JavaScript, WebGL, WebRTC, Network Adversity.*

Agradecimentos

A Deus, pelas visíveis providencias e a certeza da direção.

A minha esposa Cristiane, pela paciência, incentivos e apoio.

Aos meus pais e irmão, que mesmo geograficamente longe, apoiaram e incentivaram de forma bem presente.

Aos meus colegas de jornada Adjeryan e James, pelo companheirismo desde nossa graduação.

A Casa Publicadora Brasileira, e de forma mais direta ao gerente de T.I. Gilvan B. de Camargo e ao diretor financeiro Pr. Edson E. de Medeiros, pela confiança, ajustes realizados e apoio financeiro, que tornaram este trabalho possível.

Ao professor e orientador Dr. Marcelo de Paiva Guimarães, pela atenção, compreensão, dedicação e incentivos que começaram mesmo antes do ingresso no programa de Mestrado da Faccamp.

Aos professores Dra. Ana Maria Monteiro, Dr. Shusaburo Motoyama e Dr. Bianchi S. Meinguins, que aceitaram avaliar este trabalho, tornando-o mais excelente possível.

Ao Dr. Diego R. C. Dias, os ensinamentos técnicos, orientações e prestatividade.

À instituição Faccamp através do coordenador do programa de mestrado Prof. Dr. Osvaldo L. de Oliveira, pela dedicação ao programa.

Aos professores do corpo docente da Faccamp.

Aos funcionários da Faccamp, que sempre prezam pelo bom funcionamento do programa.

SUMÁRIO

GLOSSÁRIO

3D: Tridimensional

AG: Aglomerados Gráficos

API: *Application Programming Interface*

CAVE: *Cave Automatic Virtual Environment*

CRT: *Cathode Ray Tube*

DLP: *Digital Light Processing*

DOM: *Document Object Model*

DTLS: *Datagram Transport Layer Security*

DTLS: *Datagrama de segurança da camada de transporte*

FIFO: *First In Fisrt Out*, primeiro a chegar é o primeiro a sair

GIF: *Graphics Interchange Format*

GUI: *Graphical User Interface*

HMD: *Head-mounted display* - dispositivo de vídeo usado na cabeça como um capacete

HTML: *HyperText Markup Language*

HTTP: *Hypertext Transfer Protocol*

ICE: *Interactive Connectivity Establishment*

IETF: *The Internet Engineering Task Force*

IMS: Subsistema Multimídia IP

IP: *Internet Protocol*

ITU: *International Telecommunication Union*

JSEP: Protocolo de Estabelecimento de Sessão JavaScript

JVM: Máquina virtual java

LAN: *Local Area Network*

LCD: *Liquid Crystal Display*

MPI: *Message Passing Interface*

NAT: *Network Address Translation*

NAT: *Network Address Translation*

NetEm: *Network emulation*

NFV: *Network Function Virtualization*

OGG: *Ogging*

P2p*:* Ponto a ponto

PC: *Personal computer*

PVM: *Parallel Virtual Machine*

QoS: qualidade de serviço do usuário

RADE: *Resource-aware Distributed Browser-to-browser 3D Graphics Delivery in the Web*

RV: Realidade Virtual

SCTP: *Stream Control Transmission Protocol*

SGI: *Silicon Graphics International*

SIP: *Session Initiation Protocol*

SIP: *Session Initiation Protocol*

SO: Sistema Operacional

SSE: *Stands for Server-sent Events*

STUN: *Session Traversal Utilities for* NAT

TCP: *Transmission Control Protocol*

TLS: *Transport Layer Security*

TURN: *Traversal Using Relays around* NAT

UDP: *User Datagram Protocol*

VoIP: *Voice over Internet Protocol*

VRML: *Virtual Reality Modeling Language*

W3C: *World Wide Web Consortium*

WAV: *Waveform Audio File Format*

WebGL: *Web Graphics Library*

WebRTC: *Web Real-Time Communications*

X3D: *Extensible 3D Graphics*

XHR: *XMLHttpRequest*

XML: *Extensible Markup Language*

XMPP*: Stands for Extensible Messaging and Presence Protocol*

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1 - Introdução

Aplicações de Realidade Virtual (RV) imersivas e interativas permitem que os usuários participem de simulações de experiências da vida real, como pilotar aviões, ou mesmo situações impossíveis do cotidiano, como a visita a um mundo microscópico. Nestas aplicações os usuários são capazes de entrar em ambientes virtuais que podem ser visualizados com diferentes graus de imersão e interação. Para isso, os ambientes de RV geralmente usam dispositivos de propósito especial que podem envolver qualquer dos sentidos básicos dos seres humanos (visão, audição, paladar, olfato e tato) (Finkelstein e Suma, 2011), (Ranasinghe e Do, 2016). Assim, a RV fornece recursos que podem complementar e intensificar o processo de ensino e treinamento (Li e Buchthal, 2012), que estão cada vez mais sendo implantados em muitos contextos, como engenharia (Pedras, Raposo e Santos, 2013), química (Muhanna, 2015), simuladores de voo (Abbasi e Baroudi, 2012) e de direção automobilística (Kemeny, 2014).

As aplicações de RV têm como principais características (Guimarães, 2004), (Kirner, 1997):

- Imersão: visa permitir que os usuários tenham a sensação de fazer parte do ambiente simulado. Muitas tecnologias imersivas podem ser usadas para atingir altos níveis de imersão, como por exemplo, monitores estereoscópicos tridimensionais (3D) e dispositivos de rastreamento de movimento;
- Interatividade: tem como objetivo fazer com que o ambiente simulado responda às ações dos usuários.

As aplicações de RV geralmente exigem alto poder computacional, por isso, nos primórdios, era comum o uso de supercomputadores para atender a demanda de processamento necessária (Cruz et al., 1992), o que dificultava o avanço de pesquisas na área. Outro fator que dificultava o avanço das pesquisas era a limitação a certas plataformas de *software*, como a IRIX das estações de trabalho da SGI (*Silicon Graphics International*) (Soares, Raffin e Jorge, 2008). Contudo, os avanços de *hardware* e *software* dos últimos anos mudaram esse cenário. O uso de equipamentos *commodity*

(mercado de consumo final) e o surgimento de novas soluções de desenvolvimento fez com que ocorresse uma expansão das aplicações de RV nos últimos anos. Atualmente, os aglomerados gráficos (AG), também conhecidos como *clusters*, são considerados a arquitetura padrão para implementar ambientes de RV com alto grau de imersão e interação (Kemeny, 2014) (Bues et al., 2001) (Humphreys et al., 2001) (Humphreys et al., 2002). No entanto, como os *clusters* de computadores têm como princípio a distribuição de tarefas/processos entre várias máquinas (nós) em uma rede de computadores, que também pode ser local ou remota, ainda existem desafios a serem vencidos, principalmente, na área de rede e multiprocessamento.

Outros desafios que os pesquisadores precisam lidar para a implementação de ambientes de RV são as custosas dependências de tempo de desenvolvimento da parte gráfica destas aplicações, que atualmente são desenvolvidas utilizando soluções de alto nível, como a Unity 3D (Jackson, 2015) e Ogre3D (Kerger, 2010). Além disso, realizar porte[1] de aplicações existentes também é outro desafio.

Nos dias atuais, existe uma tendência de utilizar soluções *web* para as aplicações de RV imersivas e interativas. Isso tem ocorrido devido a evolução dos navegadores em relação ao suporte de tecnologias gráficas, como no caso de HTML5 (Jamsa, 2013) e padrões como WebGL (*Web Graphics Library*) (Danchilla, 2012) e asm.js. Essa abordagem *web* acarreta o benefício da não necessidade de instalação, configuração ou presença de aplicativos nativos a determinados Sistemas Operacionais (SO). Assim, as aplicações atingem um alto grau de portabilidade (Moreau, 2013).

Esta pesquisa tem como objetivo analisar a tecnologia WebRTC (*Web Real-Time Communications*) presente nos navegadores, como solução de comunicação ponto a ponto (p2p) para aplicações de RV *web* de multi-projeção com alto grau de imersão e interação, e que são baseadas em *clusters* de computadores. Para isso, é apresentada uma arquitetura e testes de rede que indicam a viabilidade, como baixa latência na rede,

[1] Termo originado da palavra em inglês *porting*, utilizado para descrever o processo de adaptação do código fonte de uma aplicação para que continue sendo executada, porém, com algumas alterações e sem perder suas características originais. Sendo que estas adaptações consistem em poucas interferências no código original (Soares, Raffin e Jorge, 2008), (Dias et al., 2010), (Dias, 2011), (Dias, 2016).

perda, corrompimento e reordenação de pacotes. Além disso, é apresentada uma aplicação multi-projeção que se utiliza da comunicação p2p proposta pelo WebRTC, apresentando-se como uma solução viável e um facilitador para a comunicação neste tipo de aplicação.

A Figura 1 ilustra o modelo tradicional de implantação de um sistema de multi-projeção (Froehlich e Livingston, 2014) comparado ao modelo proposto neste trabalho. No primeiro modelo (A) a implementação da aplicação é nativa ao SO, sendo assim, toda a parte de comunicação (troca de dados na rede), renderização gráfica e interação com os recursos disponíveis no SO são gerenciados pela aplicação. Nessa abordagem há a necessidade de instalação de bibliotecas específicas para o SO, o que restringe a portabilidade. O modelo B ilustra o ambiente de sistema de multi-projeção proposto neste trabalho, no qual a aplicação responde pela interpretação dos dados recebidos e sua renderização, ficando a disponibilização dos recursos do nó para a aplicação a encargo do navegador. Nesse modelo é possível obter o resultado esperado de uma aplicação de multi-projeção concentrando os esforços do desenvolvimento especificamente no ambiente virtual, sendo possível executar a aplicação em navegadores *web* compatíveis com a tecnologia requerida no desenvolvimento. Como a maioria dos SO contam com navegadores compatíveis, então essa solução é portável, pois elimina a necessidade de instalações específicas de bibliotecas.

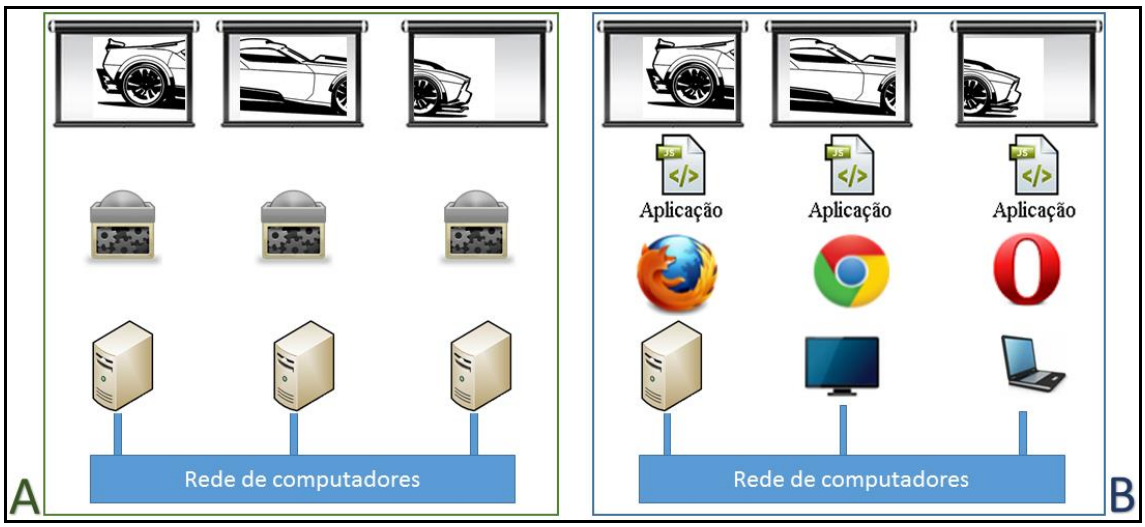

Figura 1 - Estruturas de sistemas de multi-projeção

1.1. Motivação

A RV juntamente com os sistemas de multi-projeção são uma grande contribuição para as comunidades científicas e industriais. Com o uso destas tecnologias é possível criar ambientes interativos e imersivos destinados à investigação e aprimoramento tecnológico (Guimarães, 2004). Com base nesta visão, este trabalho é motivado pelas seguintes perspectivas:

- Demonstrar a viabilidade de uso dos navegadores *web* como plataforma de execução das aplicações de RV com alto grau de imersão e interação;
- Facilitar a implementação da comunicação entre os nós de um *cluster* de computadores via a API (*Application Programming Interface*) WebRTC.

1.2. Organização da dissertação

A dissertação está organizada da seguinte forma: o capítulo 2 trata inicialmente dos sistemas de RV e foca, principalmente, nos sistemas de multi-projeção e suas características que motivam o desenvolvimento e pesquisa de novas soluções visando seu melhoramento; o capítulo 3 apresenta a tecnologia WebRTC, trazendo características de sua arquitetura de funcionamento, sua API para desenvolvimento, descrição dos protocolos que fazem parte de sua estrutura de comunicação, a sinalização que proporciona a conexão entre os nós e a técnica NAT (*Network Address Translation*) para que esta conexão ocorra. Neste capítulo também serão apresentadas as bibliotecas e servidores pesquisados para o desenvolvimento das aplicações de testes; o capítulo 4 mostra a possibilidade de integração entre WebRTC e *clusters*, voltando à tona o conceito de sistemas de multi-projeção e a apresentação da arquitetura proposta para desenvolvimento de tais sistemas. Também serão discutidas as diretrizes definidas para os testes de comunicação e estudos de caso; e, por fim, o capítulo 5 apresenta as conclusões.

Capítulo 2 - Sistemas de Multi-Projeção

A RV teve início na década de 60 e pode ser definida como uma técnica avançada de interface, na qual o usuário pode realizar imersão, navegação e interação em um ambiente sintético 3D gerado por computador, utilizando canais multissensoriais (Guimarães, 2004), (Kirner, 1997).

A RV vem possibilitando que diversas áreas experimentem simulações de ambientes, o que em situações reais seria mais custoso, perigoso ou inviável. Por exemplo, na engenharia é possível simular uma situação antes de sua efetiva construção e na medicina uma cirurgia pode ser estudada e planejada antes de sua execução (Guimarães, 2004).

2.1. Características das aplicações de RV

Aplicações de RV podem ser distinguidas em duas categorias (Pinho, 2000), (Kirner, 1997):

- **Não-imersivas**: são aplicações com baixo grau de imersão. Geralmente, utilizam equipamentos comuns, como computadores pessoais e dispositivos móveis (Kirner, 1997);
- **Imersivas**: são aplicações com alto grau de imersão. Buscam explorar todos os sentidos fundamentais dos seres humanos. Elas visam fazer com que o usuário se sinta como parte do mundo simulado. Para isso, utilizam equipamentos específicos, como HMD (*Head-mounted display* - dispositivo de vídeo usado na cabeça como um capacete) e CAVE (*Cave Automatic Virtual Environment*) (Kirner, 1997), (Pinho, 2000), (Dias, 2016), (Hoang et al., 2016), (Ranasinghe e Do, 2016).

De forma geral as aplicações de RV tendem a possuir um alto grau de imersão e interação. Para tanto, elas incorporam recursos como:

- Analogia com o mundo real: com o objetivo de aumentar o grau de imersão dos usuários as aplicações utilizam os conhecimentos prévios dos

usuários nas simulações. Por exemplo, se o usuário empurrar uma bola, ela se locomoverá (Guimarães et al., 2003), (Dias, 2016);

- Interface de alta qualidade: visando uma melhor forma de interação com o usuário final, uma aplicação busca possuir uma interface de alta qualidade, que utilize o conhecimento intuitivo que o usuário já tem no mundo físico para interagir com a interface de controle das aplicações de RV, como o movimento da cabeça para direcionar o ângulo de visão dentro da aplicação ou o movimentar da mão para manipular ou orientar elementos (Kirner, 1997), (Hoang et al., 2016);

- Resposta às interações: juntamente com uma interface que recebe os estímulos do usuário no mundo real, deve haver uma reação adequada dentro do mundo virtual. Essa interatividade necessita ser previamente programada promovendo assim o maior número possível de ações oferecidas ao usuário (Guimarães et al., 2003). Todo o cuidado e preparo da aplicação referente à interface com interação e realismo tem como objetivo representar o mundo real em alguma situação, função ou ambiente específico, atuando com o usuário de tal forma que este possa se sentir imerso neste ambiente, envolvendo os sentidos para que intuitivamente ações corriqueiras possam ser utilizadas e interpretadas pela aplicação para o controle e manipulação da realidade (Guimarães et al., 2003), (Lochtefeld, Kruger e Gellersen, 2016), (Ranasinghe e Do, 2016).

Com isso é possível destacar a interface de alta qualidade, a interatividade, a imersão, o envolvimento de sentidos e a analogia com o mundo real, como sendo características que impulsionam a imersão e a interação dos usuários nas aplicações de RV.

2.2. *Clusters* de computadores

A popularização dos *clusters* ocorreu na década de 1990 tornando possível a execução de aplicações de alto desempenho com processamento em lote. Hoje a utilização de *clusters* atinge diversas áreas, tanto científicas quanto comerciais (Guimarães,

2004), (Soares, Raffin e Jorge, 2008), (Froehlich e Livingston, 2014), (Kemeny, 2014), (Froehlich e Livingston, 2014), (Li, Kulik e Ramamohanarao, 2016).

Uma característica que contribuiu para a popularização dos *clusters* é o seu baixo custo de investimento no quesito *hardware*, pois sua construção é feita a partir de componentes de *hardware* comuns no mercado de consumo final. De forma diferente, os computadores especializados dependem exclusivamente de determinadas empresas. O acesso a equipamentos que compõem os *clusters* é garantido, já que os fabricantes estão a cada dia produzindo dispositivos e partes mais potentes (Soares, Raffin e Jorge, 2008), (Froehlich e Livingston, 2014), (Guimarães et al., 2003). Além disso, geralmente eles contam com soluções de *softwares* de código livre, o que permite personalizá-los conforme a necessidade (Soares, Raffin e Jorge, 2008), (Finkelstein e Suma, 2011).

Devido aos *clusters* de computadores serem constituídos de vários nós, sendo cada nó um computador independente com seu próprio sistema operacional, a instalação, manutenção e configuração desses nós é uma tarefa que demanda tempo e recursos, tornando-se uma dificuldade considerável para gerenciamento. Mesmo havendo soluções para o gerenciamento, como clonagem de disco rígido e distribuições de sistemas operacionais voltados especificamente para trabalhar em *clusters*, as atualizações constantes de bibliotecas e versões de *softwares* ainda tornam a manutenção dos nós um trabalho custoso (Soares, Raffin e Jorge, 2008), (Froehlich e Livingston, 2014).

2.2.1. Aglomerados Gráficos

Os Aglomerados Gráficos (AG), que são os utilizados nesta pesquisa, em síntese consistem em *clusters* de computadores voltados para o processamento gráfico. Isso significa que os nós destes *clusters*, interconectados por uma rede de dados, têm seu desempenho voltados especificamente para a renderização de imagens com alta qualidade gráfica. Enquanto que nos *clusters* convencionais as tarefas a serem executadas são divididas em sub-tarefas menores e distribuídas entre os nós para serem executadas. Já nos AG cada nó é responsável por renderizar uma parte dos dados que foram distribuídos possibilitando perspectivas diferentes de um mesmo conjunto de dados (Guimarães et al., 2003).

A utilização de AG iniciou no final da década de 1990, quando começaram a ser utilizados como uma alternativa promissora para a renderização de imagens devido ao custo-benefício, apresentando grandes vantagens, comparado aos supercomputadores, como a sua escalabilidade e menor custo (Froehlich e Livingston, 2014), (Guimarães et al., 2003). Com isso, começaram os trabalhos de desenvolvimento de *softwares* e *hardwares* voltados para serem utilizados em *clusters*. Novas técnicas de sincronização de dados, comunicação entre nós, compartilhamento e utilização eficiente de recursos permitiu o desenvolvimento de aplicações de RV cada vez mais complexas (Froehlich e Livingston, 2014).

Os nós das aplicações de RV de um AG compartilham as informações conforme a arquitetura Mestre/Escravo Com Replicação ou Mestre/Escravo Sem Replicação. A arquitetura Mestre/Escravo Sem Replicação, que pode ser vista na Figura 2, consiste em uma solução centralizada, onde somente o nó mestre executa toda a aplicação localmente, recebe e trata as interações do usuário e se comunica com os outros nós escravos para o processamento de renderização. É possível destacar as seguintes características dela (Guimarães, 2004), (Kim et al., 2015):

- Apresenta uma arquitetura de sistema transparente, tanto para o processo de desenvolvimento como para a usabilidade final, já que não é necessário implementar métodos de distribuição e sincronização de dados de entrada para todos os nós, pois estes dados são processados no nó principal;
- Melhor manutenção, já que apenas em um nó do AG é instalada a aplicação e os dispositivos de interação com o usuário, isso apoia o desenvolvimento e manutenção eficiente;
- Requer grande largura de banda da rede devido às atualizações de processamento gráfico gerado pelo servidor que são distribuídas entre os nós do AG. Isto apresenta um problema para aplicações interativas 3D, pois estas exigem constantes atualizações de quadro.

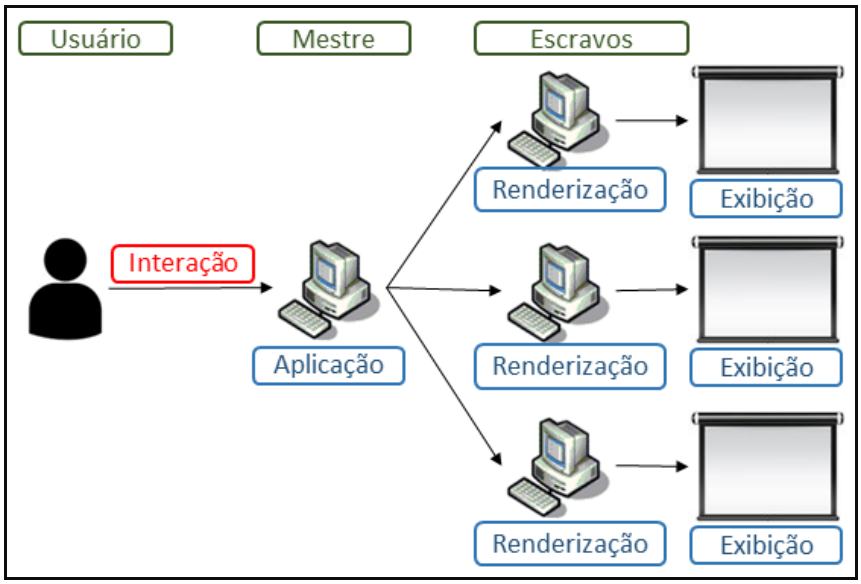

Figura 2 - Mestre/Escravo sem replicação

Na arquitetura Mestre/Escravo Com Replicação (Figura 3) todos os nós executam a mesma aplicação e somente o nó mestre recebe e distribui as informações referentes à interação com o usuário. As características desta arquitetura são (Guimarães, 2004), (Kim et al., 2015):

- Não exige uma grande largura de banda, já que os dados transmitidos na rede entre os nós são variáveis compartilhadas obtidas na interação com o usuário; isso resulta em um menor tráfego de rede comparado à arquitetura anterior, apresentando-se como uma abordagem mais adequada em aplicações interativas 3D neste quesito;
- Por ser uma abordagem mais distribuída, apresenta uma arquitetura com uma baixa transparência, provocando um aumento do custo nas fases de desenvolvimento e manutenção, pois requer a implementação de métodos de sincronização dos dados compartilhados entre os nós, garantindo assim uma consistência dos dados;
- O nó Mestre também pode ser usado para a renderização como um dos nós escravos;
- Requer alta capacidade de processamento dos nós envolvidos, pois todos executam a aplicação e geram as imagens.

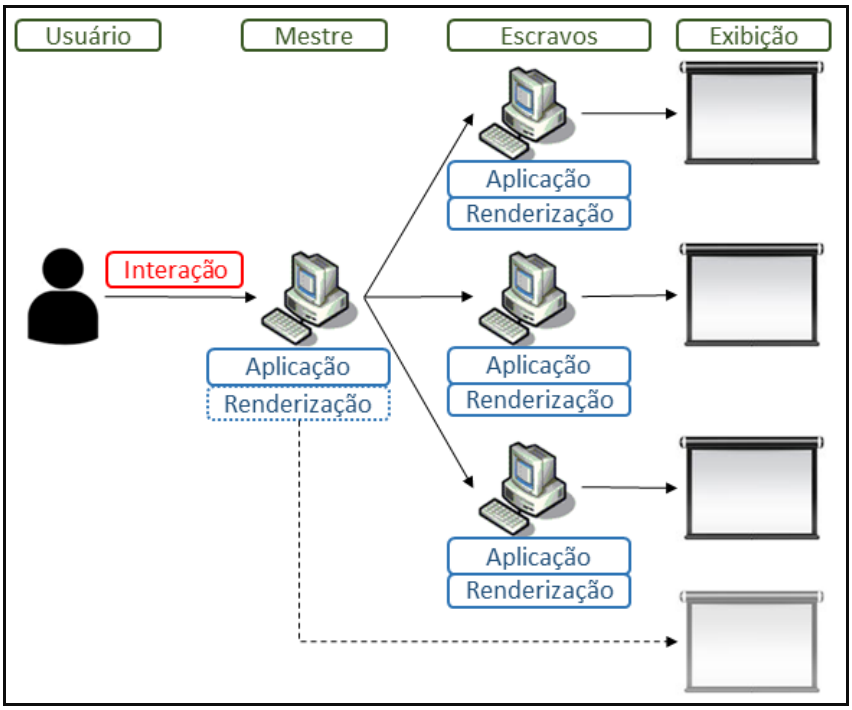

Figura 3 - Mestre/Escravo com replicação

Utilizando-se das vantagens dos AG, foi possível a pesquisadores desenvolverem sistemas capazes de estimular os sentidos dos usuários de forma multissensorial como visão e audição, e alcançar as principais características da RV que são a interatividade e imersão (Guimarães, 2004).

2.4. Sistemas de Multi-projeção

Com o objetivo de proporcionar um alto grau de imersão dos usuários nos ambientes virtuais, os sistemas de multi-projeção vêm sendo estudados a mais de uma década. Estes sistemas são compostos por múltiplas telas que vão desde monitores CRT (*Cathode Ray Tube*) a projetores de alta capacidade de resolução como os DLP (*Digital Light Processing*) e LCD (*Liquid Crystal Display*) (Zuffo, J. et al., 2001), (Guimarães, 2004), (Soares, Raffin e Jorge, 2008), (Dias, 2016).

Dentre os diversos tipos de sistemas de multi-projeção desenvolvidos, é possível hoje mencionar o *Video Wall* (parede de multi-projeção), que consiste na disposição de suas telas no formato de matriz, como pode ser visto na Figura 4 e as CAVEs (Figura 5) (Zuffo, J. et al., 2001), (Guimarães, 2004), (Soares, Raffin e Jorge, 2008). E em 2010

foi apresentado o sistema mini-CAVE (mini-caverna) (Figura 6), que oferece uma disposição de 3 telas anguladas de forma diferente das CAVEs originais, porém, reduzindo o custo de sua implementação (Dias et al., 2010), (Dias, 2016).

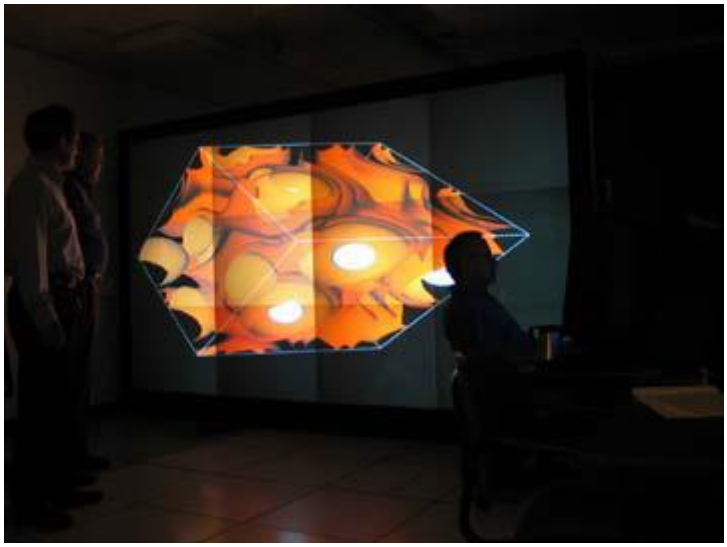

Figura 4 - *Display Wall* com suas telas dispostas no formato 3X4 (Guimarães, 2004)

Figura 5 - CAVE (Dias, 2016)

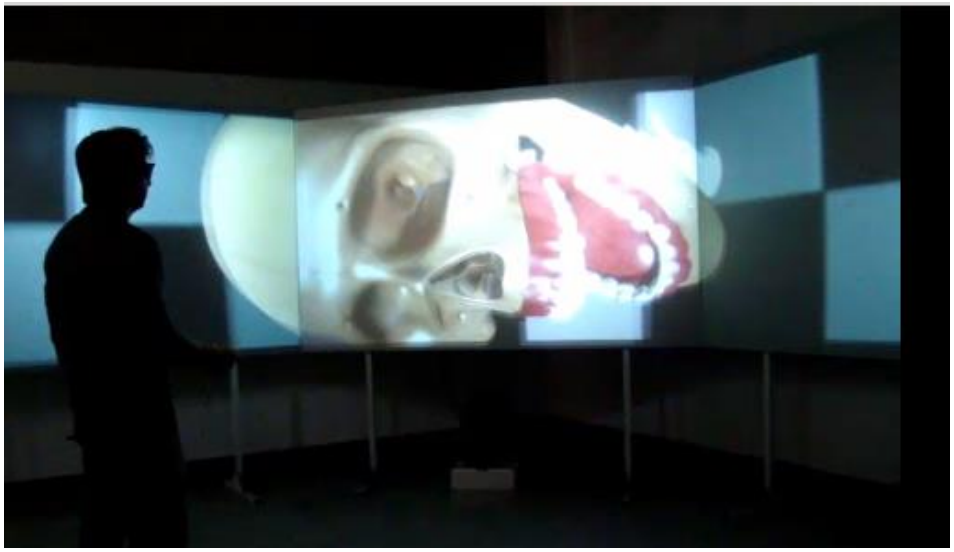

Figura 6 - mini-CAVE (Dias, 2016) apud (Dias, 2011)

As CAVEs, em especial, proporcionam uma melhor imersão aos usuários do que soluções como *VideoWall* e HMDs, pois o campo de visão do usuário é 360º graus e o usuário é inserido fisicamente no espaço virtual em vez de somente sua representação virtual ou de parte do corpo, como somente as mãos ou apenas a cabeça (Froehlich e Livingston, 2014).

A visualização imersiva que as CAVEs criam, juntamente com a interação através de dispositivos de entrada que captam os movimentos dos usuários, como, por exemplo, da cabeça ou do braço, mostram-se como as maiores vantagens que este tipo de sistema proporciona, pois permitem que o usuário interaja mudando o ponto de vista do ambiente em que está inserido de forma intuitiva, podendo assim analisar diferentes partes dos dados apresentados e com uma visão mais ampla do todo (Froehlich e Livingston, 2014).

O potencial de uso de CAVEs é relevante para os mais diversos setores, como automotivo, petrolífero, energético, aeronáutico, biociência e saúde. Para servir a tantas áreas de interesse, muitas pesquisas têm se dedicado a melhoria dos diversos pontos de gargalo, como as taxas de renderização das imagens, latência de redes e dos dispositivos de interação humana (Zuffo et al., 2001), (Froehlich e Livingston, 2014).

As CAVEs na atualidade geralmente utilizam AG para gerenciar e processar o ambiente a ser projetado. Eles permitem a construção de ambientes de RV com alto

grau de imersão e interação dos usuários (Froehlich e Livingston, 2014), (Dias, 2016). Com a necessidade crescente de apresentar imagens realistas, a geração de ambientes em tempo real exige uma demanda computacional que se apresenta como uma das maiores barreiras para a RV, pois sistemas computacionais com alto desempenho gráfico elevam o custo (Zuffo et al., 2001). Inicialmente, para alcançar um desempenho aceitável com relação a qualidade gráfica foram utilizados sistemas fortemente acoplados. Estes sistemas consistem em máquinas com alto poder de multiprocessamento, e por isso de alto custo, inviabilizando pesquisas em laboratórios com poucos recursos financeiros disponíveis (Guimarães et al., 2003), (Kemeny, 2014).

Esta inviabilidade para centros de pesquisas com menos recursos impede que outras frentes de pesquisas sejam iniciadas e contribuam para o avanço tecnológico em áreas que possam se beneficiar da RV, restringindo o desenvolvimento a grupos de pesquisas com interesses estritamente direcionados a resultados econômicos, sendo necessário observar outras possíveis soluções que trouxessem desempenhos equivalentes aos sistemas fortemente acoplados.

Com o passar dos anos foi percebido o aumento no desempenho das placas gráficas em computadores pessoais, trazendo a atenção dos pesquisadores para os sistemas fracamente acoplados, que se apresentam como uma solução mais viável financeiramente. Com isso os AG estão sendo a solução adotada em novas frentes de pesquisas voltadas às aplicações de RV (Guimarães et al., 2003), (Finkelstein e Suma, 2011), (Kemeny, 2014), (Dias, 2016).

É possível então dizer que sistemas de multi-projeção são sistemas de RV executados em um AG, que consiste na geração de várias visões de um mesmo conjunto de dados, onde cada nó/computador no aglomerado processa a visualização de cada visão.

2.6. Distribuição de dados em sistemas de multi-projeção baseados em aglomerados gráficos.

Uma das características apresentadas dos sistemas de multi-projeção baseados em aglomerados gráficos é que os nós são interconectados por uma rede de dados (seção 2.2.1. Aglomerados Gráficos). Por isso, o mecanismo de comunicação usado pelas apli-

cações para distribuir os dados deve ser implementado de forma que sejam entregues para os nós em tempo real, garantindo que a interatividade do usuário com a aplicação proporcione a imersão desejada.

Apesar de um dos fundamentos desejados nos sistemas distribuídos é não ter um ponto único de falha para nenhuma atividade, geralmente no caso das aplicações de RV algumas funcionalidades acabam sendo centralizadas em alguns nós devido às especificidades próprias. Por exemplo, geralmente existe um nó mestre responsável pela obtenção dos estímulos gerados pela interface da aplicação com o usuário. Como consequência, a distribuição consistente desses dados entre os nós deve proporcionar um processamento da visualização coerente do ambiente individual. Para atingir esta consistência dos dados entre os nós, geralmente utilizam-se os seguintes métodos de distribuição (Guimarães et al., 2003):

- Distribuição de estímulos: consiste em retransmitir o mesmo estímulo obtido pelos dispositivos de entrada para todos os nós. Cada nó trata os estímulos individualmente e geram a renderização do ambiente (Figura 7). Essa abordagem exige que a interação comece somente após todos os nós estarem em execução. Se um nó falhar, esse não poderá retornar. Isso acontece porque o cálculo do nó depende do estado anterior da aplicação.

- Cálculo dos resultados dos dados centralizados e distribuição: o nó mestre primeiramente calcula os valores do ambiente virtual necessários para a renderização e transmite estes valores para os nós, que ao receberem, renderizam a área do ambiente do qual são responsáveis com os novos valores (Figura 8). Essa abordagem fornece a flexibilidade de nós serem adicionados em qualquer momento, pois receberá o resultado já calculado conforme o estado atual da aplicação.

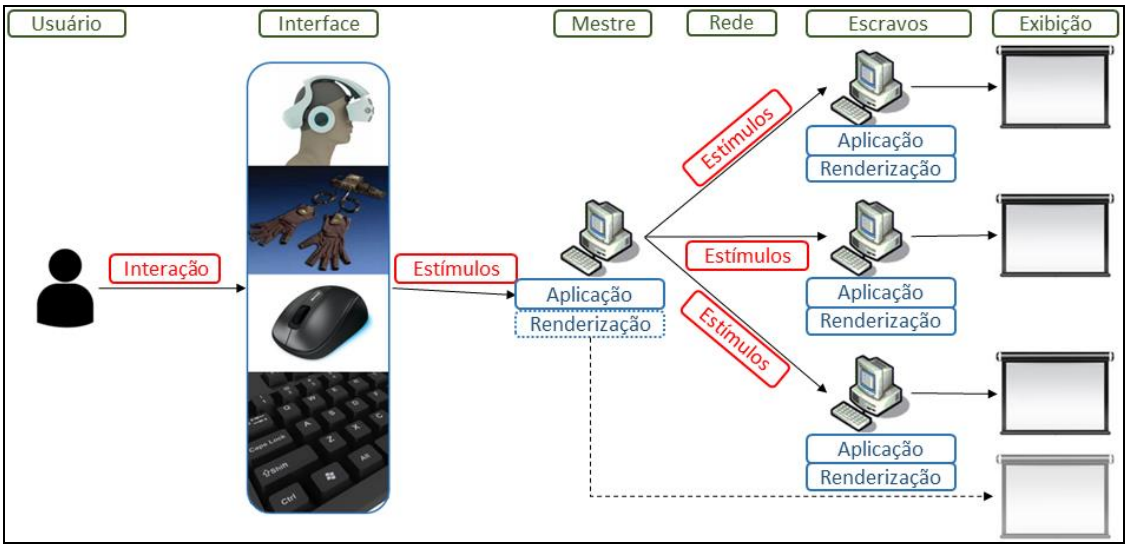

Figura 7 - Distribuição de estímulos

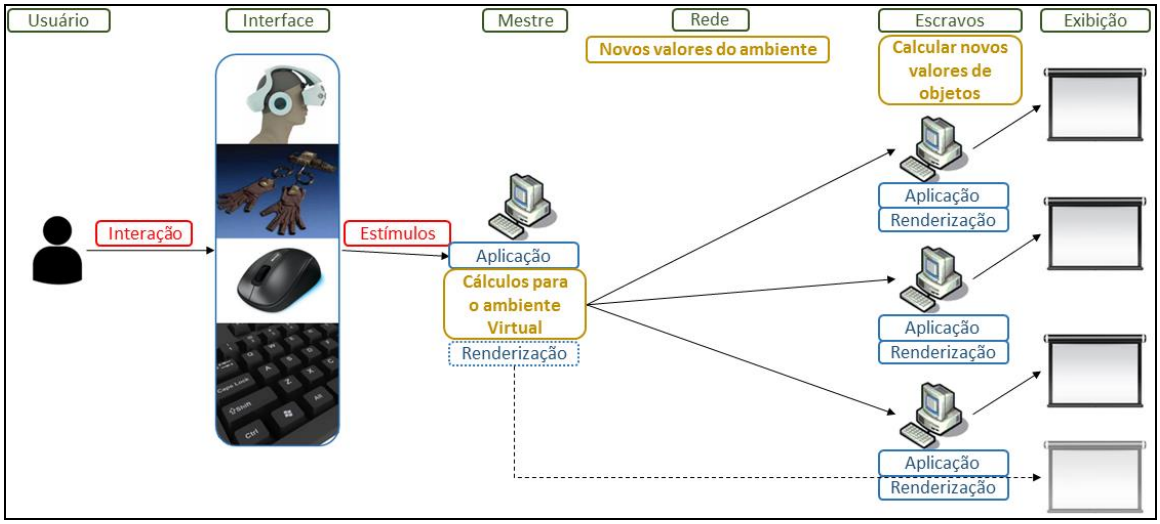

Figura 8 - Distribuição de cálculos do ambiente virtual

A distribuição dos dados pela rede ocorre através da utilização de técnicas e bibliotecas de comunicação que visam tornar a comunicação confiável e o mais ágil possível. Algumas das principais soluções disponíveis para este objetivo serão apresentadas nas seções seguintes.

2.6.1. PVM (Parallel Virtual Machine)

A PVM foi desenvolvida a partir de bibliotecas e rotinas em C e Fortran, e tem por objetivo tornar nós diferentes dentro da rede como um recurso virtualmente único (MPI, 2016). Ela permite utilizar os protocolos TCP (*Transmission Control Protocol*) ou UDP (*User Datagram Protocol*) na camada de transporte. Além disso, para resolver problemas de heterogeneidade, fornece o serviço de empacotamento de mensagens antes do envio e o desempacotamento no recebimento pelo processo que recebe. Isso gera uma carga extra de processamento.

Sendo o PVM considerado o precursor do padrão MPI (*Message Passing Interface*), hoje os maiores esforços e pesquisas se concentram neste, que está em sua terceira versão (MPI, 2016), (Nielsen, 2016).

2.6.2. MPI (Message Passing Interface)

MPI é uma API que define uma interface de troca de dados através do envio e recebimento de mensagens entre nós de um *cluster*. Esta API consiste em funções para envio e recebimento de mensagens entre os processos, podendo a comunicação ocorrer de forma ponto a ponto entre cada processo ou de forma coletiva em operações globais, como atualização de variáveis globais da aplicação (Ignácio e Filho, 2002), (MPI, 2016), (Nielsen, 2016).

Iniciado em 1991, a biblioteca MPI é uma interface de comunicação bastante difundida para a utilização em computação distribuída de alta performance. Várias linguagens de programação como C, C++, Java, Fortran e Python disponibilizam os métodos MPI de comunicação através da sua API, tornando possível a utilização destes métodos de forma nativa em cada linguagem (Ignácio e Filho, 2002), (MPI, 2016), (Nielsen, 2016).

Para operações globais a MPI dispõe de funções otimizadas que auxiliam os desenvolvedores ao lidar com algoritmos paralelos, estas funções são (Ignácio e Filho, 2002), (MPI, 2016), (Nielsen, 2016):

- *Barrier* (Barreira): sincroniza todos os processos que fazem parte do grupo MPI, impedindo que novas instruções sejam realizadas até que todos do grupo tenham passado por esta instrução de barreira;
- *Broadcast* (Difusão): envia o dado de um processo para todos os outros processos;
- *Gather* (Reunir): reúne dados de todos os processos em um único processo implementado especificamente para este fim. Um comando é disparado para todos os outros processos que responderem com seus valores ao processo criado para este fim. Desta forma é possível que a aplicação trabalhe com os valores retornados da forma como for necessário;
- *Scatters* (Disseminar): esta é o inverso da operação *gather*, distribuindo um conjunto de informações para os processos;
- *Reduce* (Redução): executa operações lógicas como soma, subtração e multiplicação nos valores dos processos, e envia o resultado para um processo específico.

A Figura 9 ilustra a ação da função *Barrier*, que obriga todos os processos a esperarem em um determinado ponto do processamento até que todos os processos do grupo tenham chegado na mesma barreira. No caso, o processo P0 concluiu suas instruções no tempo t1 e o processo P1 concluiu suas instruções no tempo t3, e ambos tiveram que aguardar a conclusão das instruções do processo P2 para que pudessem continuar com as instruções seguintes. Esta função geralmente é usada antes de um envio de mensagens que atualize todos os processos com valores necessários para as próximas instruções (Ignácio e Filho, 2002), (MPI, 2016), (Nielsen, 2016).

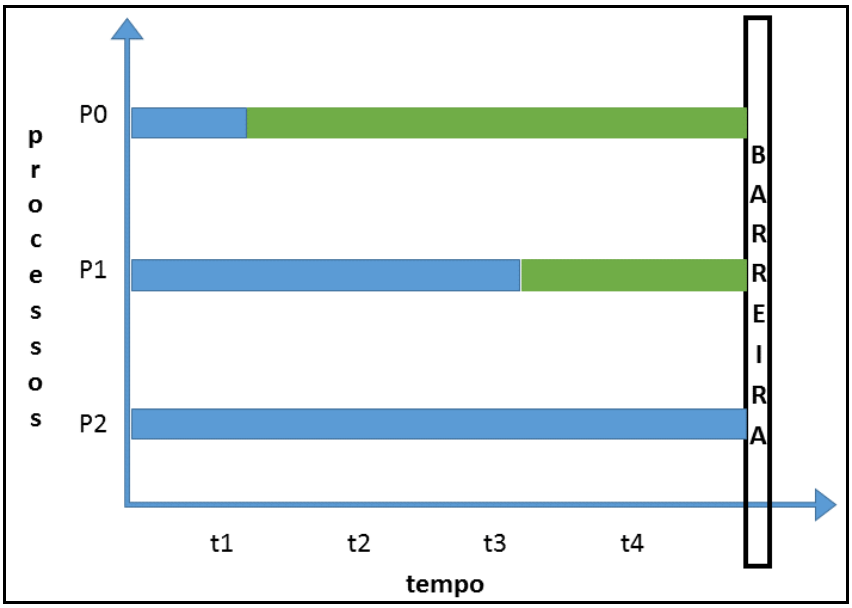

Figura 9 - Função barreira

A Figura 10 ilustra a instrução *Broadcast*, que é utilizada para enviar um mesmo dado para os processos MPI. Esta função difere da instrução *Scatters* que também envia mensagens para os processos com dado, porém transmite um conjunto de dados entre os processos (Figura 11) (Ignácio e Filho, 2002), (MPI, 2016), (Nielsen, 2016).

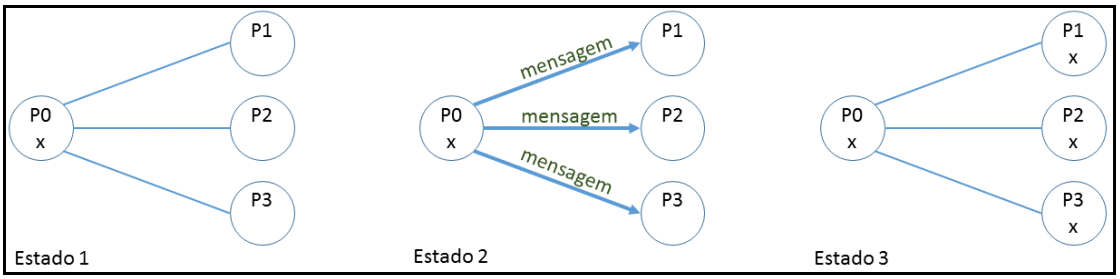

Figura 10 - Função broadcast

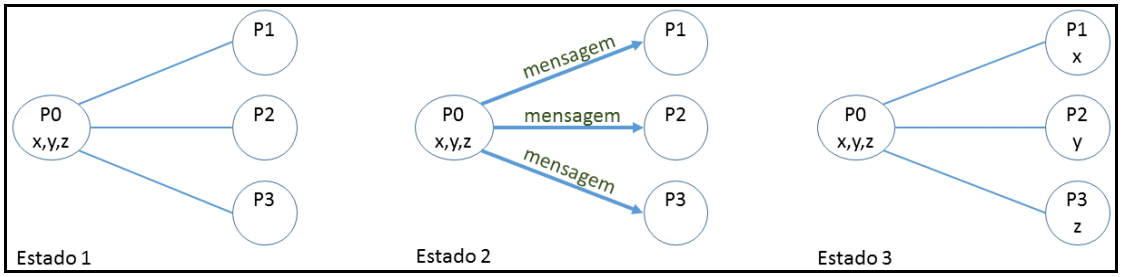

Figura 11 - Função scatters

A Figura 12 ilustra a função *Gather*, que ao contrário da função *Scatters*, recebe no processo P0 após os recebimentos das mensagens enviadas, o conjunto de dados resultante dos outros processos MPI. Este é um exemplo de instrução que pode ser executada após uma *Barrier*, garantindo que a próxima etapa esteja com valores sincronizados (Ignácio e Filho, 2002), (MPI, 2016), (Nielsen, 2016).

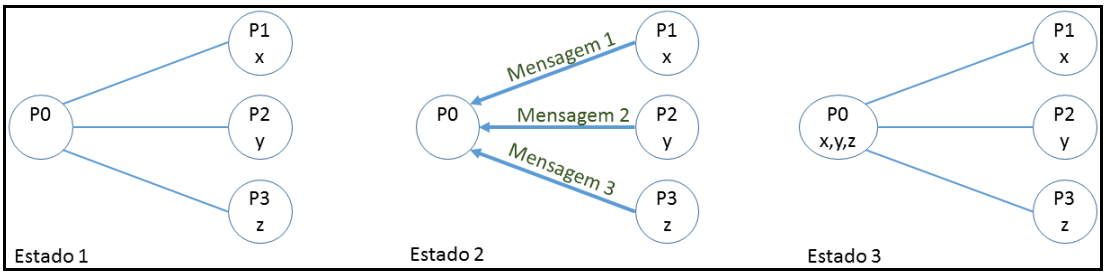

Figura 12 - Função gather

A função *Reduce* é ilustrada na Figura 13 que, neste exemplo, está aplicando uma soma de todos os valores nos processos P1, P2 e P3 que são enviados para o processo P0 (Ignácio e Filho, 2002), (MPI, 2016), (Nielsen, 2016).

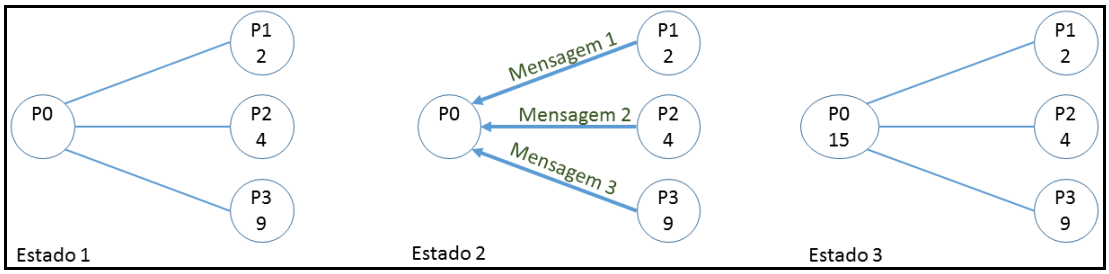

Figura 13 - Função reduce

Sendo uma das mais difundidas e estudadas bibliotecas de comunicação para implementações de aplicações para *clusters*, o padrão MPI foi apresentado com seus principais conceitos de comunicação (Ignácio e Filho, 2002), (MPI, 2016), (Nielsen, 2016).

2.6.3. LibGlass

Desenvolvida inicialmente com o nome de DICElib pelo Laboratório de Sistemas Integráveis do Departamento de Engenharia de Sistemas Eletrônicos da Universidade de São Paulo, ela consiste em um conjunto escalável de componentes que podem

ser utilizados para desenvolver aplicações de multi-projeção baseadas em AG (Guimarães, 2004), (Dias, 2016).

Na Figura 14 é apresentada sua arquitetura, que é agrupada em um arcabouço (*framework*) que abrange três componentes (Guimarães, 2004), (Dias, 2016):

- Instanciação: permite criar aplicações nas arquiteturas servidor–cliente, mestre–escravo e ponto-a-ponto;
- Protocolo: tem a função de encapsular bibliotecas de comunicação, como por exemplo a MPI e PVM, padronizando a comunicação entre protocolos como TCP, UDP e SCTP;
- *Plugins*: disponibiliza funcionalidades de transmissão de eventos, compartilhamento de dados, criação de barreiras de sincronização, associação de funções (Alíases), sincronização remota de dados, acesso concorrente sobre objetos compartilhados, transmissão ou recebimento de vídeos ou áudios e verificação de adversidades da conexão em tempo de execução.

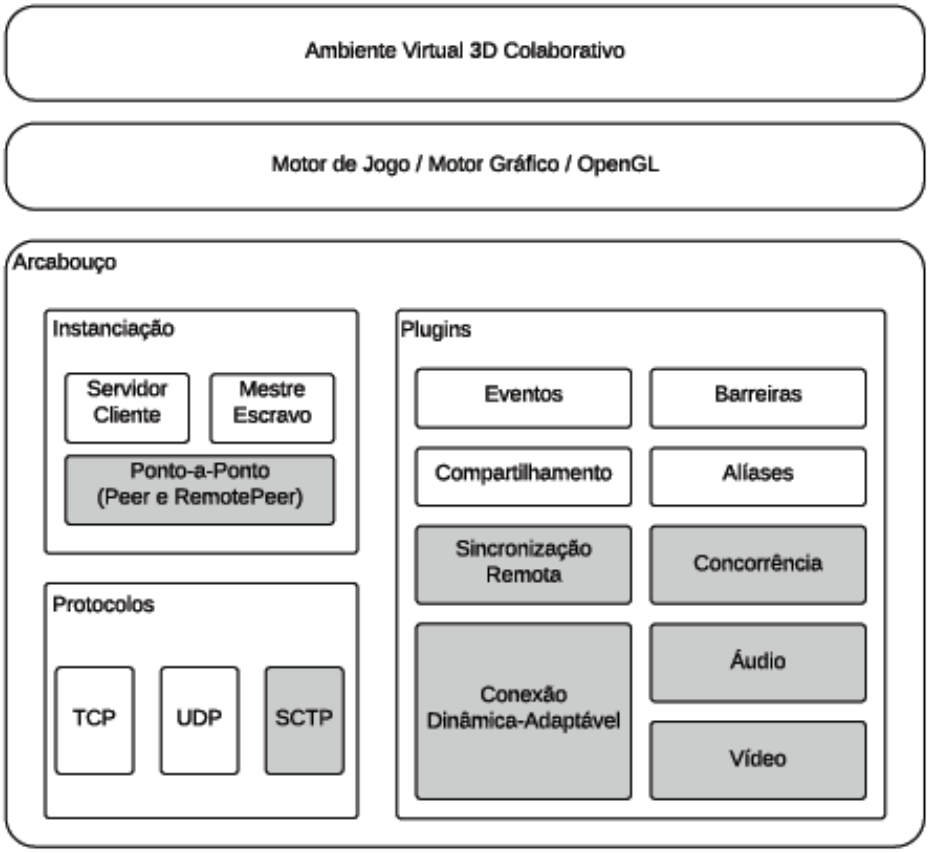

Figura 14 - Arquitetura libGlass (Dias, 2016)

Assim, a libGlass visa auxiliar o desenvolvedor com recursos de comunicação de dados, processamento distribuído, gerenciamento distribuído, acesso distribuído dos dados e de distribuição dos dados dos dispositivos no aglomerado gráfico.

2.7 Considerações finais do capítulo

Neste capítulo foram apresentados os conceitos básicos que compõem uma aplicação de RV, em seguida foi apresentado o *cluster* de computadores como uma alternativa para a renderização destas aplicações em vez dos custosos supercomputadores.

Devido aos avanços na área de *hardware* para computadores pessoais, os AGs, que consistem em *clusters* voltados para o processamento gráfico, toraram-se a melhor solução para a execução de aplicações de RV desde o final da década de 1990.

Neste contexto apresentado surgem os sistemas de multi-projeção, aplicações de RV desenvolvidas para proporcionar um alto grau de imersão utilizando-se de vários

dispositivos de renderização para a apresentação do ambiente virtual. Juntamente com os sistemas de multi-projeção, apresenta-se o desafio da distribuição de informações entre os nós, necessária para a correta execução da aplicação.

Para a distribuição dos dados nos AGs foram apresentados os métodos distribuição de estímulos e distribuição de resultados de cálculos centralizados, sendo a distribuição de estímulos o método adotado neste trabalho. Foram apresentadas algumas das principais soluções disponíveis para a distribuição dos dados, o PVM, MPI e libGlass. Nos primórdios essas bibliotecas foram utilizadas para desenvolver diversas aplicações de multi-projeção. Contudo, com o avanço do *hardware* e das tecnologias *web*, outras abordagens sugiram, dentre elas, o uso de WebRTC.

A vantagem de utilizar tecnologia *web* pura é que se ganha portabilidade das aplicações. Contudo, ainda existem dúvidas em relação ao desempenho dessa nova abordagem. Este trabalho se concentra em apresentar o WebRTC como esta alternativa promissora para o desenvolvimento de aplicações de RV com alto grau de imersão e interação.

Capítulo 3 - WebRTC

Os diversos dispositivos conectados à *web* como *smartphones*, PCs, *Web Servers* e *SmartTvs* comumente utilizam a forma de comunicação cliente-servidor. Entretanto, dependendo da aplicação, estes dispositivos também necessitam trocar informações entre si, que vão além do modelo cliente-servidor. Este modelo é utilizado por aplicações *web* como salas de bate-papo, vídeo conferência e compartilhamento de arquivos, que são desenvolvidas para a troca de informações através de um *browser* (navegador *web*) entre dispositivos diferentes. Exemplos de aplicações *web* como estas até o ano de 2009 não possuíam tecnologia de comunicação em tempo real de forma nativa, *plug-ins* instalados nos navegadores, como *flash*, possibilitavam este tipo de comunicação, que já era presente em aplicações *desktop*. Visando solucionar esta falta de funcionalidade a equipe de desenvolvimento do Chrome pesquisou e implementou componentes de comunicação em uma interface JavaScript, iniciando assim o desenvolvimento do WebRTC (Lachapelle, 2013).

No atual cenário de grande carga de comunicação entre dispositivos através de aplicações *web*, o W3C (*World Wide Web Consortium*) e o IETF (*The Internet Engineering Task Force*) juntaram forças para a definição de JavaScript, padronização das *tags* HTML5 e protocolos de comunicação necessários para prover a troca de informações entre os navegadores de nova geração (Loreto e Romano, 2015). O resultado desta padronização é a API WebRTC. Com ela os navegadores são capazes de trocar informações ou mídias com outros navegadores em tempo real de forma p2p. Esta padronização garante também a segurança de acesso para periféricos como câmeras e microfones para a troca de dados em tempo real com qualquer outro dispositivo remoto.

Assim, o suporte nativo de comunicação p2p no próprio navegador, torna o uso de WebRTC uma solução de comunicação das aplicações de RV com alto grau de imersão e interação. Torna-se então uma possibilidade, que é investigada neste trabalho.

3.1. Arquitetura WebRTC

Muitas das aplicações desenvolvidas para a *web* são concebidas visando tralharem na arquitetura de comunicação cliente-servidor (Figura 15). Neste cenário, todas as requisições de serviços como *e-mails*, banco de dados e aplicações *web* são direcionadas a um nó na rede chamado de servidor. Os nós que fazem a requisição de um serviço são chamados de clientes. Desta forma, os clientes não trocam informações diretamente entre eles, as informações precisam ser requisitadas aos servidores.

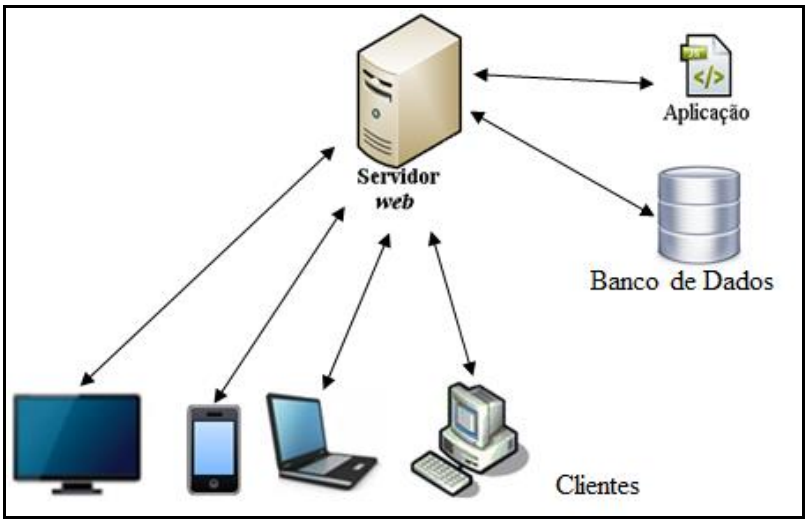

Figura 15 - Arquitetura cliente-servidor

A Figura 16 ilustra o cenário do WebRTC, que amplia a arquitetura cliente-servidor clássica. Nesse cenário utiliza-se como padrão o paradigma de comunicação p2p entre navegadores. Na arquitetura de conexão WebRTC, os navegadores executam aplicações *web*, iguais ou diferentes, que são acessadas de servidores *web*. Em cada navegador a aplicação inicia mensagens de sinalização que são transportadas via protocolos HTTP ou *WebSockets* pela *web* através de *web servers* que podem: modificar, traduzir ou gerenciá-los como for necessário. Em seguida, o canal de comunicação *PeerConnection* é estabelecido permitindo que os navegadores se comuniquem diretamente.

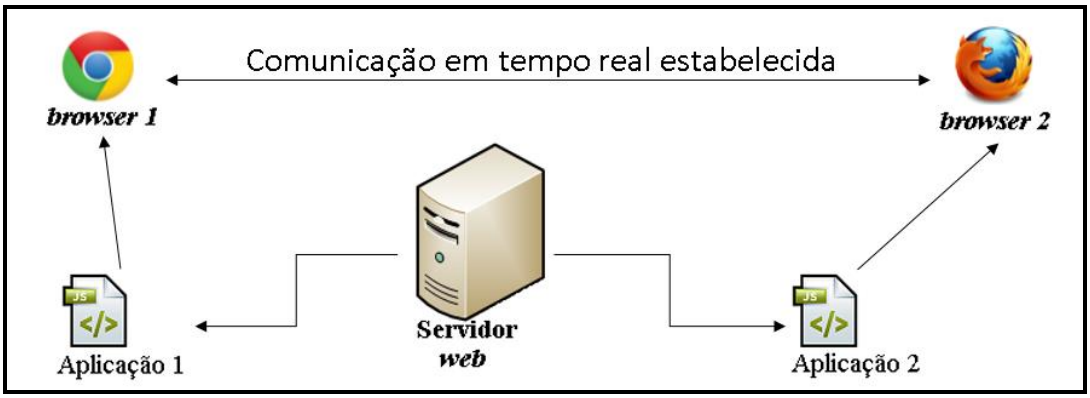

Figura 16 - Arquitetura de comunicação em tempo real

Até o momento desta pesquisa a lista de navegadores que implementaram funcionalidades WebRTC em suas novas versões é composta por: Chrome, Firefox e Opera. Já as plataformas móveis compatíveis com funções WebRTC são Android e iOS (WebRTC, 2016).

3.2. API WebRTC

A API WebRTC permite que aplicações JavaScript sejam capazes de utilizar recursos de tempo real nos navegadores como, por exemplo, áudio, vídeo, e canais de dados. Para isso, é necessário o estabelecimento de canais de comunicação entre os navegadores. Como Loreto e Romano (2015) mencionam, inicialmente a API foi projetada para três conceitos de comunicações principais: *MediaStream*, *PeerConnection* e *DataChannel*. Cada um é responsável por parte da comunicação. Cada um desses conceitos é apresentado nas subseções seguintes.

3.2.1. MediaStream

O conceito de *MediaStream* é uma representação de fluxo de dados de áudio e/ou vídeo local ou remoto, e serve para que a API gerencie e tome decisões como gravar, enviar ou receber este fluxo de dados. Quando este fluxo de mídia é originado em dispositivos como microfone e câmera local, então o fluxo de mídia é chamado pela API de *LocalMediaStream*. Este *MediaStream* local é tratado de forma diferente pela API, pois quando iniciado é disparada a diretiva de segurança que solicita a permissão do usuário para que tal fluxo seja iniciado.

Esta função de obter um recurso de mídia remoto ou local para dentro de uma aplicação abre a possibilidade para uma ampla variedade de serviços como reconhecimento facial para segurança, vídeo conferência, captura de foto, dentre outras aplicações (Levent-Levi, 2014).

3.2.2. PeerConnection

Representa o canal de comunicação de navegador para navegador entre os nós. Esta comunicação é coordenada pela aplicação que cria o canal de sinalização usando *XMLHttpRequest* ou *WebSocket*. Após o canal ser estabelecido entre os nós, os dados são enviados entre os navegadores (Loreto e Romano, 2015).

Esta é a parte central do WebRTC, pois nela encontram-se as implementações de envio e recebimento de mídia, tratamentos de problemas de rede como perda de pacotes, implementação de *codec*, NAT (*Network Address Translation*) transversal, etc (Levent-Levi, 2014).

3.2.3. DataChannel

Fornece um serviço de transporte genérico que permite uma comunicação p2p entre navegadores. O transporte de dados pode ser feito em paralelo com transporte de mídia e utilizando ainda a mesma porta sem que haja problemas nesta comunicação, pois foi idealizado para que suportasse vários fluxos de dados.

A estrutura de comunicação idealizada para o *DataChannel* atua de forma transparente para o desenvolvedor. Assim, é possível o desenvolvimento de aplicações como (Levent-Levi, 2014):

- *Streaming* de vídeo semelhante a forma que o protocolo *BitTorrent* funciona hoje;
- Compartilhamento de arquivos *on-line*;
- Aplicações de bate-papo e jogos, onde é necessária baixa latência.

3.2.4 *Codecs*

Para garantir uma linha base de interoperabilidade entre os diferentes navegadores, a IETF definiu o Opus (RFC6716) e G.711 como *codecs* de áudio obrigatórios. O Opus é um *codec* desenvolvido pela IETF para aplicações de mídia na *web* em tempo real. Com ele as aplicações podem ajustar a taxa de compressão facilmente (Lozano 2013). O *codec* G.711 é um *codec* padrão da ITU (*International Telecommunication Union*) e tem sido usado em aplicações SIP (*Session Initiation Protocol*) em tempo real (Lozano, 2013).

O VP8 e H.264 foram definidos como os *codecs* de vídeo. Estes *codecs* precisam ser implementados dentro dos navegadores para que as conversões sejam executadas (Lozano, 2013). VP8 é um *codec* de vídeo de código aberto pela Google. Ele inclui componentes que tratam a perda de pacotes, tratamento de imagens com ruído, captura e reprodução de vídeos em múltiplas plataformas (WebRTC, 2016). O H.264 é um dos *codecs* de compressão de vídeo mais utilizados. Ele é empregado em sistemas como vídeo conferência, transmissões de TV, câmeras *web*, e multimídia para portáteis (WebRTC, 2016).

3.3 Protocolos

A essência da comunicação no WebRTC é a comunicação em tempo real. Para garantir esta comunicação foi adotada uma estrutura de protocolos com a intenção de não somente garantir a pontualidade, mas também a segurança e confidencialidade.

O conceito *DataChannel* (seção 3.2.3 *DataChannel*) possibilita a comunicação p2p entre clientes. A Figura 17 descreve a utilização de tecnologias como ICE (*Interactive Connectivity Establishment*), STUN (*Session Traversal Utilities for* NAT) e TURN (*Traversal Using Relays around* NAT) encapsuladas no protocolo UDP (*User Datagram Protocol*) fornecendo assim uma solução de NAT (*Network Address Translation*) transversal, que garante a entrega na rede de dados. Já a segurança e criptografia são garantidas pela utilização do protocolo DTLS (*Datagram Transport Layer Security*), e o controle do fluxo e ordenação dos pacotes é garantida pela adoção do protocolo SCTP (*Stream Control Transmission Protocol*) (Grigorik, 2013), (WebRTC, 2016).

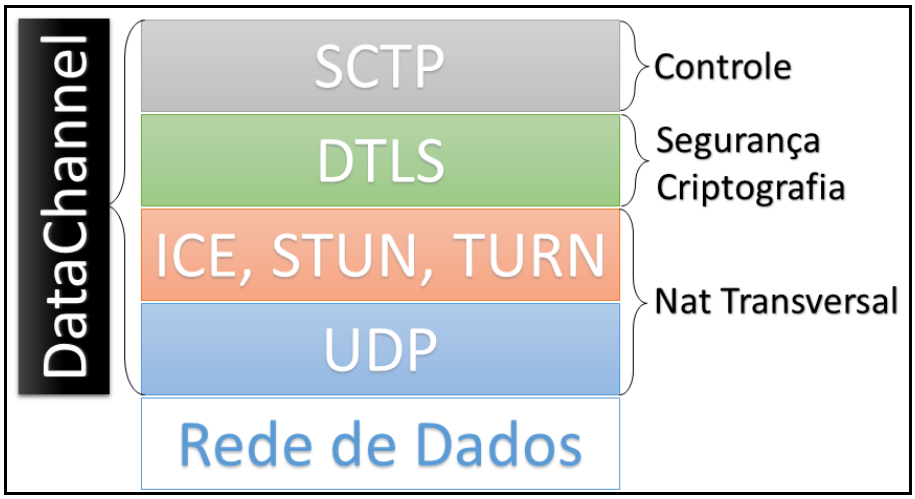

Figura 17 - DataChannel e protocolos

Cada protocolo apresentado na Figura 17 é descrito nas subseções seguintes. Os recursos que garantem o NAT transversal serão descritos na seção de NAT.

3.3.1. UDP

O UDP é usado em muitos cenários onde a comunicação em tempo real é essencial. Por isso, foi adotado como o protocolo de transporte para o WebRTC (Grigorik, 2013), (WebRTC, 2016). A proposta do protocolo UDP não inclui controle/correção de erro, controle de fluxo ou retransmissão de mensagem. Caso algum erro seja identificado, a mensagem é simplesmente descartada. Isso o torna rápido e eficiente para comunicações em tempo real (Costa, 2007).

3.3.2. Datagrama de segurança da camada de transporte (DTLS)

Na especificação do WebRTC é requerido que todos os dados, fluxo de áudio ou vídeo, sejam transferidos com criptografia entre os usuários. Uma alternativa para a implementação desta funcionalidade seria o protocolo TLS (*Transport Layer Security*), que hoje é usado em aplicações HTTPS, se não fosse sua incompatibilidade com o protocolo UDP por falta de funcionalidades como ordenação e entrega confiável de pacotes (Grigorik, 2013). Desta forma foi adotado o protocolo DTLS, que consiste na estrutura do protocolo UDP, porém acrescido de segurança (WebRTC, 2016). O DTLS é usado no WebRTC para garantir a segurança dos dados durante o tráfego na rede usando crip-

tografia, porém não trata atrasos na aplicação, perda ou reordenação de pacotes (Amaral, 2013).

3.3.3. Stream Control Transmission Protocol (SCTP)

Desenvolvido com a finalidade de troca de mensagens telefônicas na rede, o SCTP garante a entrega dos dados entre os usuários sem erros e sem duplicações, podendo retransmitir os dados quando necessário e com habilitação a tolerância a falhas de rede através do suporte a caminhos múltiplos (Costa, 2007).

Este protocolo é usado pelo WebRTC para a entrega de pacotes através do *DataChannel*. Ele pode ser considerado um protocolo híbrido pois contém características do UDP e TCP, pois quando adotado na estrutura do *DataChannel*, incorpora características do TCP que o UDP por si só não tem, tornando a entrega dos pacotes do *DataChannel* confiável em cima do transporte UDP caracterizado por sua baixa latência na rede. As características incorporadas são (WebRTC, 2016):

- Ordenação de pacotes;
- Como no TCP, o SCTP é orientado a conexão;
- Oferece controle de fluxo como o TCP, evitando congestionamento de rede.

3.4 Sinalização

O principal objetivo do projeto WebRTC foi especificar a forma de controlar a parte de mídia de uma aplicação, deixando assim que a parte de sinalização seja especificada pela aplicação (Loreto e Romano, 2015). Diferente de como ocorre em aplicações HTTP padrão, no qual o servidor só envia informações ao navegador cliente como resposta a uma solicitação, as aplicações WebRTC estabelecem uma ligação bidirecional com o servidor. Para isso, na aplicação são definidos o meio de transporte e o protocolo de sinalização.

Para o transporte há 3 opções principais:

- XHR (*XMLHttpRequest*): por vezes, referido pelo nome de Comet, é usado para a criação de *sites* interativos onde a atualização de páginas é feita de forma modular e não completa (WebRTC, 2016);
- *Stands for Server-sent Events* (SSE): estabelece o canal de comunicação com o cliente quando este abre uma solicitação. Ao invés do servidor concluir esta requisição com a resposta, é enviada a resposta sem nunca terminar, mantendo assim o canal com o cliente aberto (WebRTC, 2016);
- *WebSocket*: é a forma mais comum de estabelecer um canal bidirecional entre cliente e servidor. Geralmente é iniciado como uma solicitação de conexão HTTP, esta solicitação identifica se ambos os lados possuem suporte a *WebSocket*, se a identificação é positiva, é habilitado o canal bidirecional sobre a conexão HTTP entre cliente e servidor (WebRTC, 2016). Este será o meio de comunicação usado nas aplicações desenvolvidas para este trabalho.

No processo de sinalização é estabelecida a sessão para a troca de dados entre as duas pontas do canal. O IETF estabeleceu o JavaScript Protocolo de Estabelecimento de Sessão (JSEP) para a troca de informações das sessões locais e remotas para os aplicativos. A descrição da sessão é a informação mais importante necessária para a troca de informações, pois é nela que está especificado o tipo de mídia, formato e todas as configurações para ser estabelecido o caminho da mídia (*media path*) (Loreto e Romano, 2015).

As opções para protocolo de sinalização são as seguintes:

- *Stands for Extensible Messaging and Presence Protocol* (XMPP): usado em aplicações de mensagens instantâneas (WebRTC, 2016).
- *Session Initiation Protocol* (SIP): usado em soluções VoIP. Foi adaptado para que o WebRTC pudesse manipular as mídias descritas em sua sessão (WebRTC, 2016).

3.5. Network Address Translation (NAT)

Situado normalmente entre uma rede privada (LAN - *Local Area Network*) e uma rede pública (*Internet*), o NAT é uma tecnologia incorporada em dispositivos de roteamento como *firewalls*, *switches*, roteadores e aparelhos DSL com a função de traduzir endereços de IP privados em IP públicos e vice-versa.

Devido as aplicações WebRTC enviarem e receberem pacotes de forma p2p utilizando o protocolo UDP como transporte, estas aplicações precisam ser capazes de interpretar e encaminhar os pacotes através de redes internas e externas utilizando-se de mecanismos conhecidos como NAT Transversal (WebRTC, 2016). Esse mecanismo em WebRTC é realizado pelas tecnologias ICE (*Interactive Connectivity Establishment*), STUN (*Session Traversal Utilities for* NAT) e TURN (*Traversal Using Relays around* NAT) que serão descritas a seguir.

3.5.1. STUN e TURN

O protocolo STUN permite que uma aplicação ou nó verifique através do servidor NAT na rede o endereço e porta pública atual da conexão (Loreto e Romano, 2015). Este protocolo serve de auxílio para o ICE identificar e compartilhar o endereço público local no processo de estabelecimento de conexão com os outros nós clientes WebRTC.

O protocolo TURN permite que um nó em uma rede com um servidor NAT descubra o endereço público de retransmissão. Assim, é possível enviar e receber pacotes através da *Web* (Loreto e Romano, 2015). Servindo também de auxílio ao protocolo ICE, este é um recurso usado somente quando não é possível encontrar um caminho até o destino pelo protocolo STUN. Este requer um servidor específico disponível na rede pública para que o ICE retransmita o pacote (Amaral, 2013).

A retransmissão de pacote onera tanto em poder computacional como em largura de banda da rede e por este motivo servidores configurados para responder solicitações TURN normalmente não são encontrados de forma pública como os servidores para STUN, precisando ser configurados e mantidos especificamente para a aplicação que irá utilizá-lo (WebRTC, 2016).

3.5.2. ICE

O ICE é um protocolo para NAT transversal e, juntamente com o STUN e TURN, provê informações sobre a topologia da rede para o navegador. Assim, é possível encontrar o melhor caminho para a transmissão dos dados para o outro navegador (destino) na *Web* através dos servidores *Web* (Loreto e Romano, 2015).

Este protocolo mapeia IPs pela rede até o destino e vice-versa e em seguida verifica a conectividade até obter sucesso. Este processo de mapear endereços IPs e testar a conectividade pode levar muito tempo. Para resolver isso foi desenvolvida uma otimização no protocolo ICE, chamada ICE *Trickle*, que consiste em tornar todo o processo ICE paralelo. Ao invés de esperar primeiramente a coleta de todos os IPs mapeados, o teste de conectividade é realizado simultaneamente, podendo gerar múltiplos caminhos de sucesso com o destino (WebRTC, 2016).

3.6. Implementações WebRTC

Nesta seção são descritas bibliotecas disponíveis que facilitam o desenvolvimento de aplicações que usam os recursos da tecnologia WebRTC, dando prioridade à função de compartilhamento de dados entre as aplicações, que será a principal função utilizada neste trabalho. Também são apresentadas as opções de servidores para publicar páginas com códigos WebRTC, que ficarão disponíveis para serem acessadas pelos navegadores.

3.6.1. Bibliotecas

Com o objetivo de auxiliar o desenvolvimento de aplicações *web*, atualmente existem diversas bibliotecas que implementam WebRTC. Estas bibliotecas têm a função de abstrair grande parte da configuração necessária para a utilização das funções WebRTC disponíveis nos navegadores. Como cada empresa que desenvolveu seu navegador pode implementar chamadas de funções específicas para sua plataforma, uma aplicação dificilmente é compatível com todas as versões de navegadores disponíveis ao mesmo tempo, o que reduz a portabilidade esperada em aplicações *web*.

A W3C tem trabalhado juntamente com as empresas para estabelecer um padrão nas chamadas de funções de um modo geral. A Tabela 1 apresenta uma comparação entre as chamadas de funções segundo o padrão W3C e dois dos principais navegadores. Então, a mesma funcionalidade é chamada via assinatura de funções diferentes.

Tabela 1 - Diferenças de chamadas de funções entre Chrome e Firefox (WebRTC, 2016)

Padrão W3C	Chrome	Firefox
getUserMedia	webkitGetUserMedia	mozGetUserMedia
RTCPeerConnection	webkitRTCPeerConnection	mozRTCPeerConnection
RTCSessionDescription	RTCSessionDescription	mozRTCSessionDescription
RTCIceCandidate	RTCIceCandidate	mozRTCIceCandidate

A Tabela 2 mostra a comparação do comportamento da função de captura do fluxo de mídia disponível em 3 navegadores em suas versões de fevereiro de 2013. Porém a cada nova versão e o amadurecimento da implantação da tecnologia em seus códigos fonte, os navegadores vem adotando os padrões estabelecidos pela W3C. Um exemplo disso é o não requerimento de prefixo do fornecedor (mostrado na primeira linha da Tabela 2) nas novas versões do Firefox.

Tabela 2 - Comparativo do comportamento do fluxo de mídia (Davis e Nyman, 2013)

Comportamento	Firefox 18	Opera 12	Chrome 24
Requer prefixo do fornecedor	Sim (moz)	Não	Sim (*webkit*)
Iniciado com atributo *autoplay*	Não	Sim	Sim
Requer habilitação pelo usuário	Sim	Não	Não
Disparo de evento *playing*	Frequentemente	Uma vez	Uma vez
Suporta protocolo *file*://	Sim	Sim	Não
Notificação de transmissão	Nenhum	Ícone	Ícone animado
Solicitação de permissão	Cada carregamento da página	Somente primeiro carregamento da página.	Cada carregamento da página

3.6.1.1. rtc.io

O rtc.io é uma coleção de módulos desenvolvidos em node.js e disponíveis em *JavaScript* também, que simplificam a utilização de funções WebRTC como: acesso a câmera e microfone local, chamadas multimídia através de navegadores, configuração de canal de dados entre navegadores e configuração de servidor de sinalização. No desenvolvimento de aplicações com o rtc.io é recomendável a utilização de outro módulo node.js, o *browserify*, que é responsável por gerenciar as dependências da aplicação por outros módulos escritos em node.js que venham a ser necessários (Rtc.io, 2016).

Para estabelecer a conexão entre os clientes, é utilizado o módulo *rtc-quickconnect*, que juntamente com o módulo de sinalização garante o estabelecimento do canal de dados e de mídia. Como toda aplicação WebRTC, as aplicações que utilizam o rtc.io também precisam de um servidor de sinalização. O módulo responsável pela camada de sinalização é o *rtc-signaller*, mecanismo que funciona sinalizando os nós e coordenando as conexões estabelecidas em canais de comunicações construídos de forma bidimensional para envio de mensagens (Rtc.io, 2016).

Outro módulo disponível é o *rtc-tools*, que é composto por uma gama de ferramentas que ajudam na configuração de tarefas que devem ser executadas continuamente em aplicações WebRTC. O módulo *rtc-core* é responsável por tornar possível a interação do rtc.io com outras bibliotecas disponíveis no lado do cliente, auxiliando a utilização de funções *cross-browser* do WebRTC por exemplo (Rtc.io, 2016).

Rtc-mesh é o módulo que juntamente com outro módulo do node.js, o *Scuttle-Butt*, fornecem uma estrutura de dados sincronizados entre os nós através do canal de dados estabelecido. O *rtc-sharedcursor* trabalhando juntamente com o *rtc-quickconnect*, fornecem uma maneira prática para a implementação de compartilhamento do cursor do *mouse* entre aplicações, através de pequenos pacotes de dados de 48*bits* para todos os nós conectados (Rtc.io, 2016).

Outros módulos como *rtc-captureconfig*, *rtc-videoproc* e *rtc-audioproc* auxiliam respectivamente na configuração da captura de mídia local, processamento do vídeo e áudio no cliente. O módulo que gera e gerencia o servidor de sinalização é o *rtc-*

switchboard. Este trabalha juntamente com o *rtc-signaller*, porém do lado do servidor que hospeda e provê o serviço (Rtc.io, 2016).

Embora o rtc.io traga uma vasta quantidade de módulos que facilitem o desenvolvimento de aplicações simples e objetivas, como para mensagens instantâneas e conferências com vídeo, em aplicações mais específicas, como a que este trabalho se propõe a apresentar, a grande quantidade de módulos e suas interações e dependências uns dos outros não é apresentada de forma muito clara (Rtc.io, 2016).

A documentação se restringe a exemplos práticos, mas de compreensão dedutiva. No momento que o desenvolvedor se propõe a expandir as funcionalidades, pode-se encontrar em um ambiente com uma grande quantidade de ferramentas, dependentes umas das outras, aumentando a complexidade da aplicação em vez de simplificar seu desenvolvimento usufruindo das funcionalidades do WebRTC (Rtc.io, 2016).

3.6.1.2. SkylinkJS

Esta biblioteca facilita a utilização das funções de áudio, vídeo e transferência de dados com WebRTC. Sua implementação simples, através da inclusão da biblioteca em *JavaScript* na aplicação, disponibiliza ao desenvolvedor funções de controle de sessão/sala entre os clientes (*joinRoom*(), *leaveRoom*(), *lockRoom*() e *unlockRoom*(), *refreshConnection*()), controle do *hardware* de multimídia como vídeo e microfone, habilitando ou desabilitando o mesmo (*on*(), *off*(), *disableVideo*(), *muteStream*(), *sendStream*(), *shareScreen*(), *getUserMedia*()), e funções específicas para a troca de mensagens entre os clientes comumente disponíveis em salas de bate-papo *on-line* (*acceptDataTransfer*(), *cancelDataTransfer*(), *getUserData*(), *sendBlobData*(), *sendMessage*()). As explicações detalhadas de todas estas funções fazem parte da documentação (Temasys, 2016).

Com esta biblioteca não é necessário preocupar-se com implementação do servidor de sinalização, que normalmente é a etapa do desenvolvimento de uma aplicação em WebRTC onde mais requer conhecimentos técnicos. Através do cadastro

no *site*[2] da empresa responsável pelo desenvolvimento desta biblioteca, é possível adquirir um código de registro que será usado pela aplicação para utilizar o servidor de sinalização da empresa. Este cadastro pode permanecer no formato gratuito, onde estão disponíveis apenas funções básicas do WebRTC, contudo apresenta limitações na quantidade de clientes conectados, ou o formato pago, que habilita funções específicas da biblioteca.

Testes iniciais com esta biblioteca foram úteis para a compreensão das funcionalidades básicas do WebRTC e como uma aplicação deve interagir com tais funcionalidades.

3.6.1.3. EasyRTC

Considerada a melhor ferramenta WebRTC na primeira conferência WebRTC Expo de 2012, o EasyRTC é uma plataforma robusta e de fácil implementação, especialmente quando se trata do servidor de sinalização (Priologic Software, 2016).

As bibliotecas dessa ferramenta estão disponíveis via pacotes do node.js. Assim, a instalação e controle de dependências podem ser realizadas sem a demanda de outras soluções específicas. A documentação consiste principalmente em exemplos *online* disponíveis para *download* e implementações adicionais. Faz parte da documentação um *script* que faz toda a configuração do servidor de forma automática.

A partir do estudo do *script* de servidor do EasyRTC, foi possível a implementação do servidor que habilita a comunicação entre os nós clientes da aplicação que este trabalho propõe, pois este encontra-se disponível de forma aberta, e não consiste em um servidor já em funcionamento aguardando as solicitações de sinalização como em outras bibliotecas.

3.6.1.4. PeerJS

Esta biblioteca abstrai as chamadas de funções do WebRTC. Para criar um canal de dados é utilizado o comando *connect*. A partir dele, passando como parâmetro o

[2] https://console.temasys.io/login

código identificador do cliente remoto, o cliente local se conecta ao cliente remoto, habilitando assim os comandos que serão usados para os tipos de comunicações específicas como (PeerJS, 2016):

- *send*, envia qualquer tipo de dado como objetos, texto e arquivos;
- *call*, envia uma solicitação de estabelecimento de canal de *stream* usado em chamadas de vídeo e áudio.

Estes comandos quando executados invocam o método *on* no cliente remoto, que deve ser tratado e interpretado dependendo do comando e do tipo de requisição feita. Esta ferramenta requer que o desenvolvedor crie um cadastro para que adquira uma chave que será usada no momento de inicialização da aplicação. A chave habilita a utilização do servidor de sinalização. Atualmente, o cadastro oferece até 50 conexões concorrentes gratuitamente (PeerJS, 2016).

O servidor desta biblioteca é chamado PeerServer e está hospedado nos servidores da empresa DigitalOcean[3]. A partir desta informação, foi possível realizar testes de sinalização entre nós geograficamente distantes, fora de um LAN local, com a aplicação proposta neste trabalho, utilizando os servidores da empresa mencionada (PeerJS, 2016).

3.6.1.5. RTCMultiConnection

Esta biblioteca faz parte de um repositório com várias ferramentas desenvolvidas pelo desenvolvedor Muaz Khan chamado WebRTC *Experiment*. Neste repositório encontram-se as seguintes bibliotecas (Khan, 2016):

- RecordRTC.js: usada para gravar fluxos de mídia de áudio/vídeo com WebRTC entre navegadores;
- Translator.js: oferece suporte para o reconhecimento de voz e tradução do Google;

[3] https://www.digitalocean.com/

- RTCMultiConnection.js: fornece funcionalidade voltada para a interconexão das aplicações;
- getScreenId.js: extensão usada em navegadores da Google para a captura de tela;
- Conversation.js: habilita funcionalidades equivalentes como troca de mensagens, vídeo conferência e compartilhamento de arquivos para a aplicação que a utilizar;
- DataChannel.js: biblioteca voltada para troca de mensagens em massa entre os vários clientes de uma aplicação;
- DetectRTC.js: biblioteca de código fonte reduzido, facilitando seu rápido carregamento, que fornece recursos para a detecção da presença das funcionalidades WebRTC no navegador cliente e identificação dos dispositivos de mídia presentes;
- getMediaElement.js: permite criar os elementos/objetos HTML que representarão os dispositivos de vídeo/áudio que serão utilizados pela aplicação em algum momento;
- ffmpeg.js: habilita o processamento e transcodificação de vídeo/áudio em fluxo de mídia WebRTC no próprio navegador nos formatos WAV, OGG, WebM e MP4;
- FileBufferReader.js: permite o compartilhamento de arquivos ponto a ponto. Esta biblioteca converte o arquivo original no formato binário *array-buffer* e nos clientes finais converte do *array-buffer* para seu formato original;
- MediaStreamRecorder.js: permite gravar o fluxo de dados do áudio ou vídeo da máquina local. Além disso, fornece recursos para gravar a própria tela do navegador. O resultado final é armazenado em arquivo local nos formatos GIF, WAV e WebM;

- getStats.js: fornece funções estatísticas que retornam para a aplicação o *status* da conexão do cliente atual, sua banda de rede utilizada, perda de pacotes, endereço local e remoto, porta usada e tipo de conexão atual.

A RTCMultiConnection, desenvolvida em JavaScript, trabalha com as funcionalidades da API RTCPeerConnection do WebRTC estabelecendo cenários de multi-seções para as aplicações que a utilizar. A documentação dela oferece exemplos de troca de mensagens entre clientes, compartilhamento de arquivos, *streaming/broadcast* de vídeo, compartilhamento de tela e conversação por áudio, todos beneficiando-se da funcionalidade de estabelecimento do canal de comunicação WebRTC pela biblioteca.

O servidor de sinalização é implementado na plataforma Node.js e utiliza a biblioteca socket.io para a identificação e inicialização da conexão entre os clientes. Ela é compatível com todos os navegadores que suportam WebRTC, incluindo navegadores de dispositivos móveis.

Devido a praticidade de utilização das funções de comunicação WebRTC, a grande quantidade de exemplos e por ser uma solução completamente gratuita, esta foi a biblioteca escolhida para o desenvolvimento das aplicações deste trabalho. Estas aplicações farão uso do servidor de sinalização escrito a partir do *script* do EasyRTC.

3.6.2. Servidores

Como visto no conceito de sinalização (seção 3.4), não faz parte do WebRTC a especificação da sinalização entre os nós. Também foi visto que a forma mais comum de disponibilizar esta sinalização, que dará início ao canal bidirecional entre cada nó, é através de servidores HTTP utilizando *webSockets*. Nesta seção são apresentas duas soluções para a construção de servidores.

3.6.2.1. Vert.x

Inicialmente chamado Node.x, o Vert.x é um projeto mantido pela Eclipse Foundation. Ele tem como uma das suas principais características ser uma plataforma multi-linguagens, podendo ser usado com Java, JavaScript, Groovy, Ruby e Ceylon (Eclipse Foundation, 2016). Outra característica do Vert.x é seu comportamento de ope-

ração orientado a eventos não bloqueantes. Os servidores desenvolvidos com ele podem lidar com uma grande quantidade de concorrências usando *threads* do *kernel* (Eclipse Foundation, 2016).

O Vert.x roda em uma máquina virtual java (JVM). Ele foi desenvolvido com um núcleo pequeno, apenas 650kb. O Vert.x é leve e rápido para responder solicitações, comparado com outros *frameworks*. Ele apresenta uma boa escolha para a criação de serviços de grande performance e leves (Techempower, 2016).

3.6.2.2. Node.js

O Node.js foi desenvolvido no modo de operação orientado a eventos não bloqueantes. O Node.js utiliza um núcleo JavaScript chamado V8, que foi desenvolvido pela Google. Aplicações desenvolvidas em Node.js dispõem do módulo HTTP, que foi projetado para trabalhar com respostas em baixa latência via *Internet*. As requisições que chegam a este serviço, são tratadas de forma assíncronas, possibilitando que tal aplicação possa ser escalável (Node.js Foundation, 2016).

A escalabilidade em Node.js é possível devido a seu modo de operação, onde as requisições ou eventos que ocorrem, entram no ciclo de eventos da aplicação para serem tratados. A criação de processos filhos ocorre através de módulos como child_process.fork() e *cluster* que habilitam *sockets* (canais de comunicação) entre si, permitindo balanceamento de carga sobre os núcleos (Node.js Foundation, 2016).

3.6.3. Aplicações com WebRTC

Com o surgimento de várias bibliotecas, também vêm surgindo várias aplicações baseadas nas funcionalidades do WebRTC. Nas próximas seções serão apresentadas aplicações que se beneficiaram de alguma destas funcionalidades, mostrando as possibilidades de utilização do WebRTC em várias áreas, e que trouxeram consistentes contribuições quanto aos conceitos de comunicação p2p do WebRTC.

3.6.3.1. e-Health

Seguindo a tendência de como as tecnologias da *web* estão sendo aplicadas no setor da saúde, proporcionando serviços muito eficazes à pacientes que por várias razões têm dificuldades para viajar para hospitais, a fim de ter apoio médico, Pierleoni et al. (2016) propuseram um serviço inovador.

A proposta consiste em um serviço de vídeo aconselhamento baseado na tecnologia WebRTC que permite que qualquer pessoa, independentemente do local, possa interagir com profissionais qualificados. A ideia inicial é a fusão entre dois importantes campos de aplicação em telemedicina, que são tecnologias *telecounseling* (paciente para consultas médicas via videoconferência) e monitorização remota dos sinais vitais (Pierleoni et al., 2016).

Atualmente, muitos dispositivos estão disponíveis para os pacientes e podem ser facilmente utilizados, desenvolvendo um serviço através do qual o médico pode monitorar a condição de pacientes, usando vários sensores, e podendo falar com eles em chamada de vídeo enquanto estes estão confortavelmente em casa. Foi implementado a transmissão em tempo real e a visualização de sinais e parâmetros adquiridos pelos sensores biomédicos para o dispositivo pessoal do paciente através do RTCDataChannel (Pierleoni et al., 2016).

O serviço é capaz de estabelecer uma chamada de vídeo entre médico e paciente, para transmitir em tempo real o sinal cardíaco e a frequência cardíaca instantânea usando um Eletrocardiógrafo de 3 eletrodos e compartilhar arquivos como resultados dos testes. Independente de *softwares* ou *hardwares*, sendo necessário apenas um dispositivo comum de acesso à *Internet* (como, PC, *smartphone, tablet*, etc.) e um navegador *web* compatível com os padrões principais. A Figura 18 mostra a interface em funcionamento (Pierleoni et al., 2016).

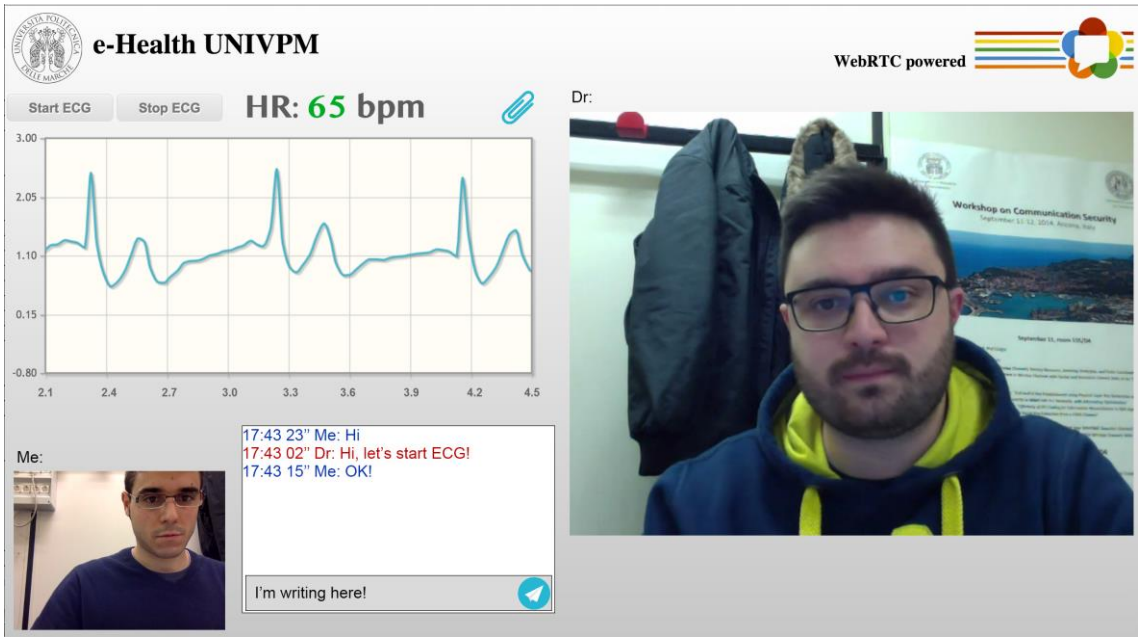

Figura 18 - Interface do e-Health (Pierleoni et al., 2016)

3.6.3.2. Tele-Board

Outra área de aplicações *web* que também é beneficiada pelo WebRTC são as aplicações de videoconferência, que tem fins tanto comerciais como educacionais. O Tele-Board consiste em um sistema de colaboração remota em tempo real que fornece um espaço de trabalho compartilhado juntamente com a funcionalidade de videoconferência. Desenvolvido para ser compatível com os padrões atuais da *web*, não depende de tecnologias de *plug-ins* proprietários (Wenzel e Meinel, 2016).

Um dos diferenciais em comparação com outras aplicações existentes hoje é que este é um dos primeiros a propor um padrão completamente compatível com o navegador *web* que suporta uma configuração de vídeo de corpo inteiro. Esta perspectiva do outro usuário, cria uma sensação de co-presença, reduzindo a necessidade de descrições verbais mais diretas. Outra vantagem é que os fluxos de áudio e vídeo são tratados por um servidor central onde é possível gravar as sessões de vídeo em um local central, algo que se fosse desenvolvido puramente em p2p seria mais complexo e exigiria um esforço muito maior no desenvolvimento. O Tele-Board fornece uma interface do histórico do navegador dando a oportunidade de controlar a linha do tempo, permitindo uma visão de trabalho assíncrona e de melhor compreensão (Wenzel e Meinel, 2016).

Para isso, foi adotada uma arquitetura diferente do padrão p2p do WebRTC, sendo implementada uma topologia em estrela (cliente-servidor) entre os clientes e o servidor principal, que grava e repassa o fluxo de áudio e vídeo. Esta solução melhora a eficiência da largura de banda da rede independentemente do número de participantes, uma vez que os participantes se conectam somente no servidor central, em vez de ser linearmente relacionado com o número de membros em conferências habituais no WebRTC. A Figura 19 exemplifica esta arquitetura adotada e seu funcionamento (Wenzel e Meinel, 2016).

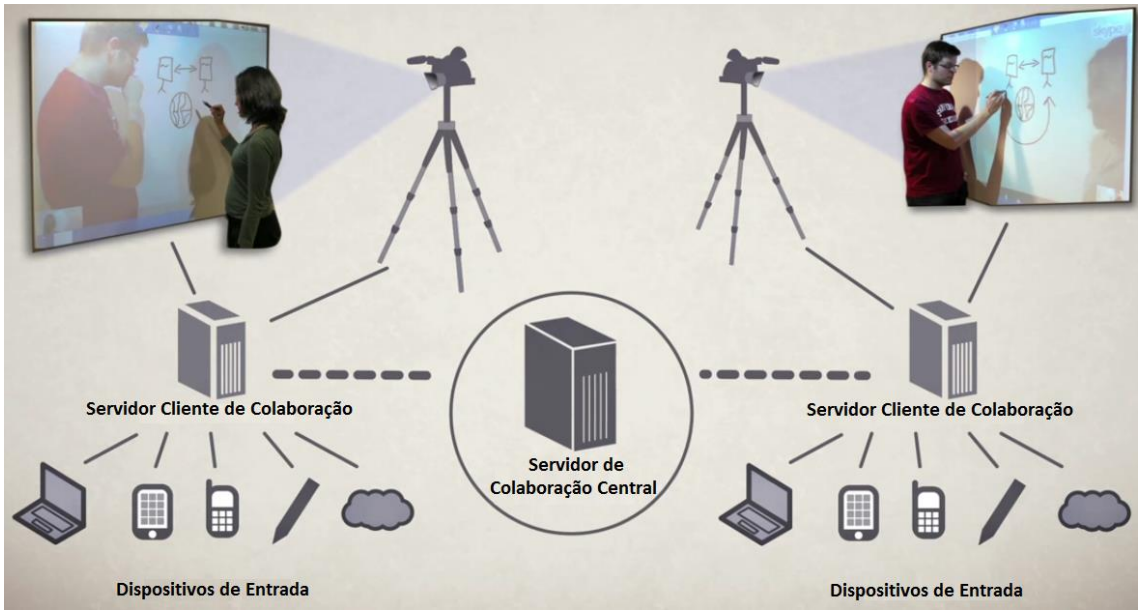

Figura 19 - Tele-Board e sua arquitetura (Hasso-Plattner-Institut, 2017)

3.6.3.3. RADE

Visando um melhor desempenho em ambientes virtuais desenvolvidos para executarem em navegadores, como os desenvolvidos em WebGL, Koskela (2015) realizou testes com o método RADE (*Resource-aware Distributed Browser-to-browser 3D Graphics Delivery in the Web*), este método permite a entrega distribuída de gráficos 3D na *web* entre navegadores de forma p2p utilizando WebRTC (Koskela et al., 2015).

A arquitetura de comunicação do método RADE usada por Koskela (Koskela et al., 2015) é apresentada na Figura 20. Nesta Figura temos a representação de clientes de um ambiente virtual que implementa sua renderização em um navegador *web*. Cada

cliente pode se conectar ao ambiente virtual por intermédio de um gerenciador de cliente, no exemplo da Figura 20 o ambiente virtual gerencia os seus clientes conectados através de dois gerenciadores. Cada gerenciador tem a função de agir como um *proxy* de sincronização responsável por manter o estado do ambiente virtual consistente entre os clientes. Este também tem a função de servidor de sinalização WebRTC/STUN, através do qual estabelece a comunicação entre os clientes provendo a entrega de recursos 3D de forma p2p entre os navegadores clientes (Koskela et al., 2015).

A comunicação entre os gerenciadores de clientes é implementada pelo DHT (*distributed hash table*) que mantém uma tabela distribuída de identificação dos gerenciadores e recursos 3D dos seus clientes (Koskela et al., 2015).

Os recursos 3D consistem em geometria, texturas, materiais de superfície e animações, que formam alguma entidade 3D, por exemplo, um edifício em uma cidade 3D virtual (Koskela et al., 2015).

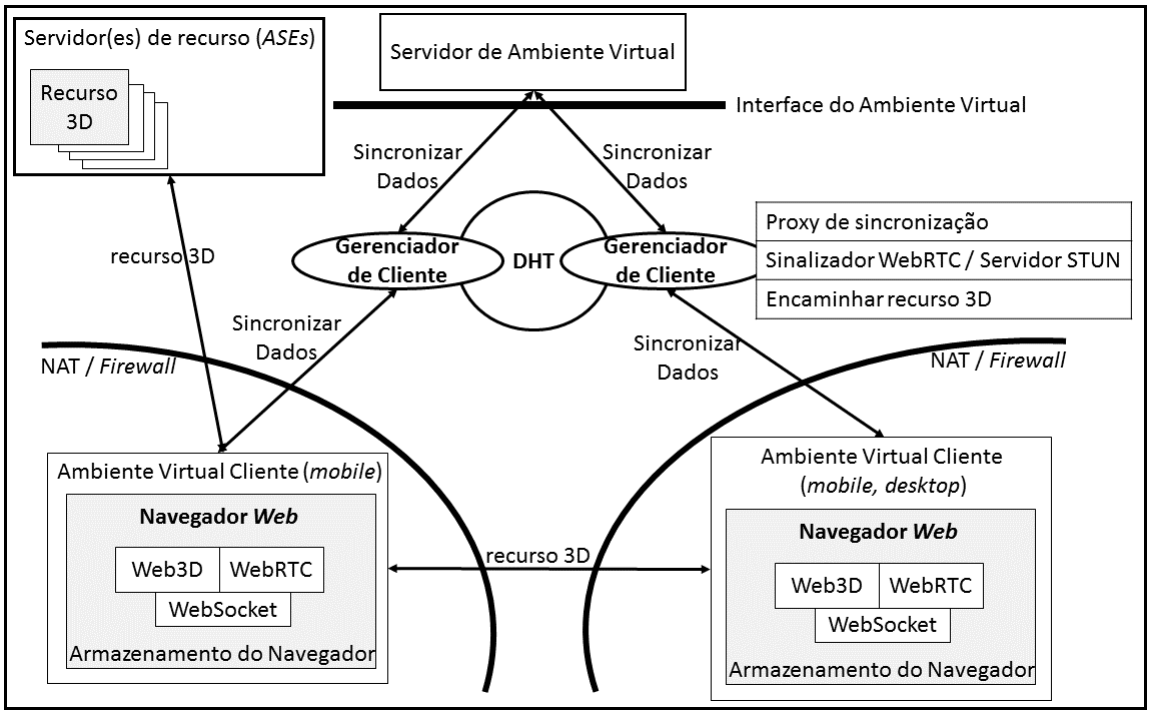

Figura 20 - Arquitetura de comunicação global para a entrega de recursos 3D adaptável e escalável, baseado em Koskela et al. (2015)

Testes foram realizados com o objetivo de confirmar que o uso da comunicação p2p na entrega de recursos 3D é benéfica pois diminui a carga sobre os servidores de

recursos 3D, diminui o tempo de resposta da aplicação devido à diminuição do fluxo de requisições, e consequentemente reduz o custo operacional da execução de todo o ambiente virtual (Koskela et al., 2015).

Com base nos resultados, o RADE pode reduzir significativamente a carga do servidor e os tempos de entrega do recurso 3D, especialmente quando os servidores de recursos 3D estão sob carga pesada. Por exemplo, com a escassa largura de banda do servidor de 2Mbps, o uso do RADE resultou em 55% do tempo de entrega dos recursos em média (Koskela et al., 2015).

Embora RADE tenha sido desenvolvido para entrega de recursos 3D, é aplicável para muitos tipos de aplicações *web*, incluindo entrega de vídeo. Como WebRTC é desenvolvido especificamente para a entrega de dados em tempo real, pode-se supor que RADE será eficaz no contexto da entrega de vídeo também (Koskela et al., 2015).

3.6.3.4. Arquitetura NFV

Já para soluções computacionais na nuvem, Nguyen propôs uma arquitetura de rede de tempo real que interliga redes WebRTC e IMS (Subsistema Multimídia IP). A tecnologia NFV (*Network Function Virtualization*) emergiu como uma solução promissora para otimizar a implantação de elementos de rede no ambiente de computação em nuvem, tanto em termos de qualidade de serviço do usuário (QoS) quanto de alocação de recursos. Para oferecer serviços avançados de IMS em várias redes de acesso, um modelo baseado em nuvem provavelmente melhora não apenas a flexibilidade no gerenciamento de rede, mas também na chamada de serviços (Nguyen et al., 2016).

Este modelo é particularmente adequado para a interligação de redes entre o IMS e o domínio de comunicação em tempo real do WebRTC, que é uma combinação natural para expandir de forma significativa potenciais pontos finais de sessões multimídia. Foi pesquisado sobre uma arquitetura NFV para permitir uma comunicação eficaz entre usuários IMS e WebRTC que suportam serviços comunitários inteligentes e proposto um modelo de otimização para projetar e alocar recursos para esse sistema, garantindo simultaneamente o nível de QoS desejado (Nguyen et al., 2016).

A arquitetura, ilustrada na Figura 21, é composta por dois domínios, WebRTC e IMS. Ambos são compostos por servidores instanciados como máquinas virtuais em *data centers* na nuvem (Nguyen et al., 2016).

Do lado do WebRTC o servidor *web* processa solicitações HTTP normais enquanto o servidor STUN/TURN é responsável por fornecer passagem NAT para conexão entre usuário WebRTC e usuário IMS (Nguyen et al., 2016).

O *Gateway* desempenha dois papéis, como um ponto final WebRTC e um cliente IMS, integrando os protocolos de comunicação do WebRTC com IMS SIP (*Session Initiation Protocol*) estabelecendo uma ponte entre os dois domínios. Isso permite que mensagens vindas de usuários WebRTC sejam enviadas para os usuários IMS e destes sejam traduzidas e transmitidas para o lado WebRTC. Desta forma é estabelecido o canal por onde os pacotes de mídia são transmitidos (Nguyen et al., 2016).

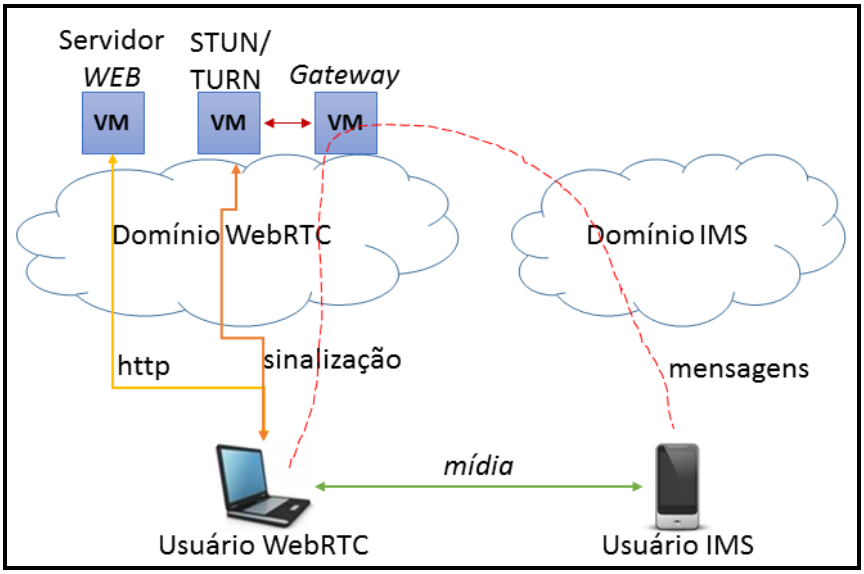

Figura 21 - Arquitetura NFV, baseado em Nguyen et al. (2016)

Esta arquitetura permite que o provedor onde os servidores estão alocados respondam de forma fácil e rápida às novas demandas de tráfego no servidor, aumentando sua capacidade ou reduzindo quando necessário. Outra vantagem também é que inovações que surjam nas tecnologias WebRTC ou IMS possam ser adotadas independentemente no *hardware*, resultando em melhores serviços aos clientes com custo monetário reduzido/customizado (Nguyen et al., 2016).

3.7. Considerações finais do capítulo

Neste capítulo foi mostrado que é necessário um mecanismo de *websocket* para o estabelecimento da sinalização inicial entre os nós das aplicações WebRTC. Embora este tipo de comunicação inicial necessite de um servidor que disponibilize este serviço para os nós, ele não é necessário durante todo o resto da comunicação entre os nós, pois os nós estabelecem suas comunicações diretamente entre si.

O Vert.x apresenta maior performance comparado ao Node.js e ambos disponibilizam funcionalidades comparáveis. Este trabalho adotará o Node.js devido à maior disponibilidade de códigos e módulos compatíveis com bibliotecas JavaScript WebRTC encontradas em seus repositórios de código aberto.

Também foi visto que a tecnologia WebRTC favorece o desenvolvimento de aplicações em várias áreas, auxiliando especialmente no quesito de comunicação em tempo real. E por ser uma tecnologia padronizada pela W3C e IETF, tem havido frequentes contribuições da comunidade científica nos testes e provas de conceitos desta tecnologia.

Os exemplos de aplicações demonstram a diversidade de áreas em que o WebRTC tem sido utilizado, o que abre a possibilidade também de ser adotado em aplicações de RV. Diante desta possibilidade, no próximo capítulo será investigada a adoção do WebRTC nas aplicações de RV, em especial as de multi-projeção, que possuem demandas específicas, como, por exemplo, renderização de imagens consistentes entre os nós em tempo real.

Capítulo 4 - WebRTC em Sistemas de Multi-Projeção

Com base nos conceitos sobre sistemas de multi-projeção e WebRTC apresentados nos capítulos anteriores, este capítulo apresenta a arquitetura de *software* criada que permite às aplicações de multi-projeção utilizarem navegadores *web*. Esta arquitetura tem como principal característica a portabilidade, pois utiliza-se de tecnologias como HTML5 e WebGL. Com isso, os detalhes específicos dos sistemas operacionais são abstraídos, o que torna as aplicações multi-plataforma.

Inicialmente o capítulo apresenta os trabalhos correlatos. Em seguida, detalha a arquitetura criada. Logo após, considera e analisa esta arquitetura diante do desafio da utilização de AG, principalmente em relação a comunicação dos nós na rede baseada na utilização do WebRTC. Para isso, são apresentados os resultados dos testes de desempenho realizados. Por fim, é mostrado um estudo de caso.

4.1. Trabalhos correlatos

Vários esforços foram realizados para investigar a utilização de arquiteturas baseadas na *web* para executar aplicações de RV. Esses esforços investigaram soluções proprietárias que requerem *software* específico. Soares e Zuffo (2004) apresentaram um navegador modificado, denominado JINX, que suporta ambientes de multi-projeção e uma série de dispositivos interativos. Este navegador é baseado em X3D (*Extensible 3D Graphics*), que é um formato de arquivo padrão e arquitetura em tempo de execução para representar e comunicar cenas e objetos 3D usando XML (*Extensible Markup Language*). O navegador JINX pode ser executado em distribuições Linux e no IRIX. A Figura 22 apresenta o navegador em um ambiente multi-projeção com monitores CRT, neste exemplo de execução é apresentado um modelo virtual de uma casa.

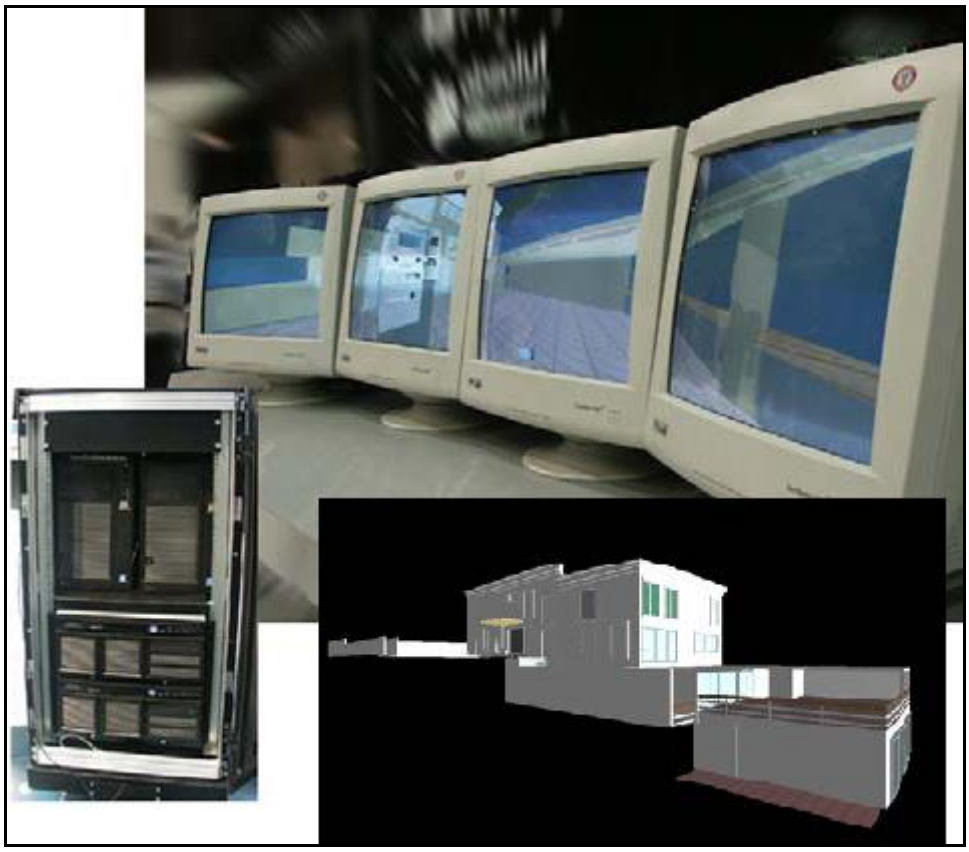

Figura 22 - JINX renderizando modelo de casa em ambiente multi-projeção (Soares e Zuffo, 2004)

Isakovic et al. (2002) descreveram um navegador *web* VRML97 modificado para suportar sua solução de sincronização. A solução deles usa um *cluster* de computador e VRML (*Virtual Reality Modeling Language*), que é um formato de arquivo padrão para representar gráficos vetoriais 3D interativos como pode ser visto na Figura 23 Kuntz (2015) descreveu o MiddleVR *toolbox*, que suporta multi-projeção e *cluster*. Com ele, os desenvolvedores criam aplicativos que podem ser publicados em plataformas de *software* como iOS, Android, Windows, Mac OS, na *Web* e plataformas de *hardware*, como consoles e CAVEs. No entanto, a versão da *web* depende de instalação de *plug-in*. Consequentemente, as melhorias desta ferramenta e novos recursos dependem da estratégia da empresa. A Figura 24 mostra o ambiente de configuração do MiddleVR.

Figura 23 - X-Roms (Isakovic et al., 2002)

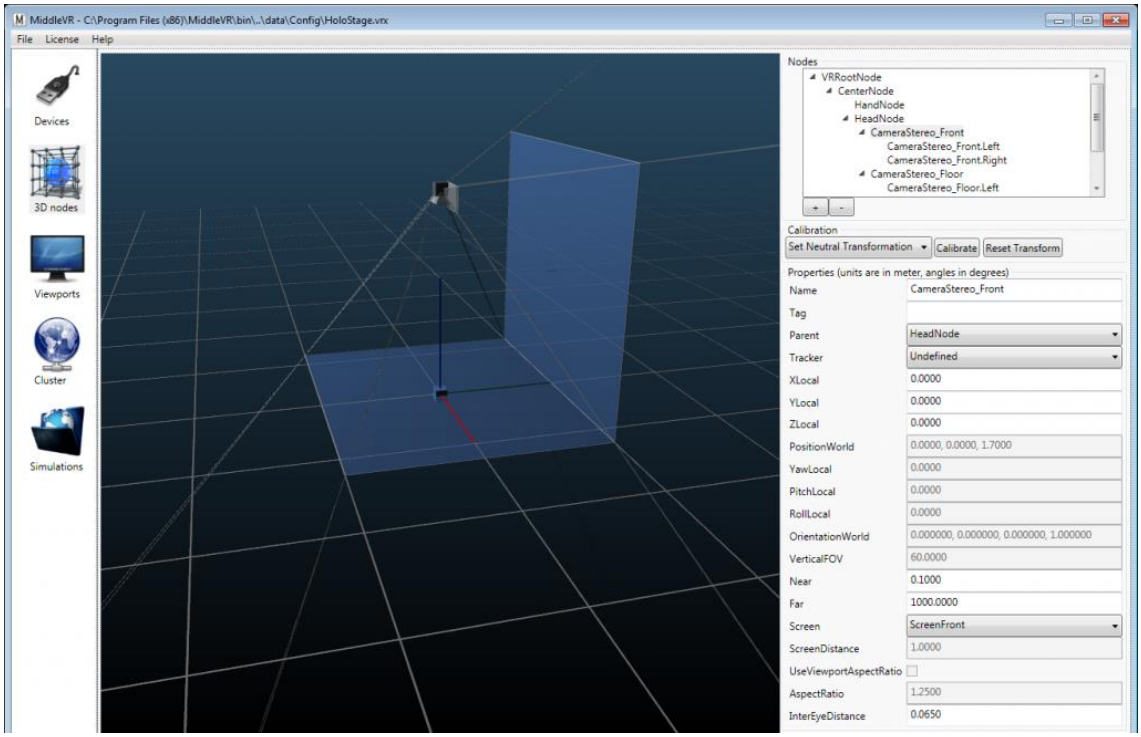

Figura 24 - Ambiente de configuração MiddleVR (MiddleVR, 2016)

Neto (2015) criou um conjunto de componentes de arrastar e soltar que permite que aplicativos Unity 3D sejam executados em *clusters* com suporte passivo para estereoscopia, correção de perspectiva de acordo com o ponto de vista dos usuários e acesso a servidores especiais para fornecer recursos independentes do dispositivo. Esta solução também pode ser executada em um navegador da *web*. Para este fim, no entanto, um *plug-in* deve ser instalado. Além disso, a versão atual não suporta objetos animados. A

Figura 25 mostra o jogo StarTrooper, que pertence ao pacote de demonstração do Unity, adaptado com os componentes para rodar em uma mini-CAVE.

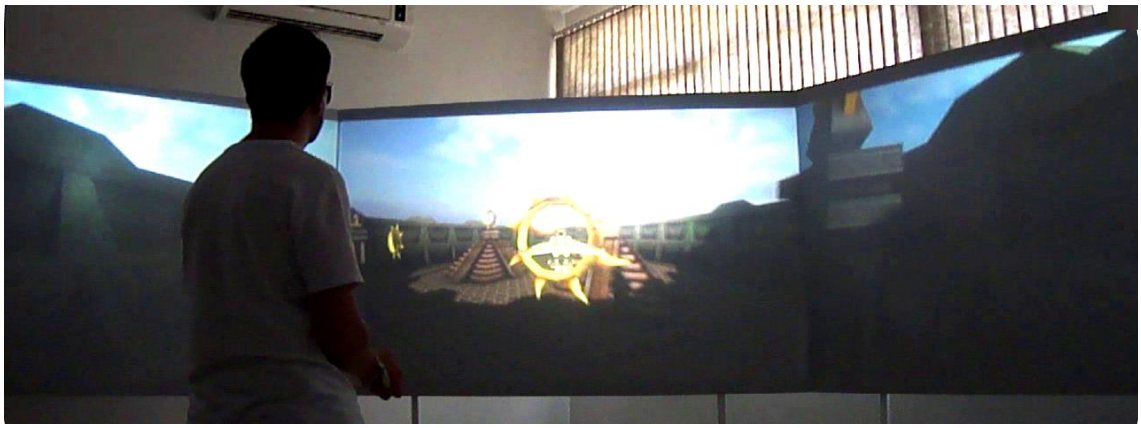

Figura 25 - Jogo StarTrooper rodando na mini-CAVE (Neto, 2015)

Kim et al. (2015) em sua solução utilizam *cluster* para a renderização de aplicações desenvolvidas em HTML e X3DOM. Desta forma a aplicação RV se torna multi-plataforma, podendo ser executada em qualquer sistema que rode um navegador *web*. Na Figura 26 pode ser vista sua solução sendo executada em um VideoWall e nos navegadores de dispositivos móveis, mesmo de configurações diferentes. O X3DOM foi padronizado pela W3C e consiste na integração do X3D (*Extensible 3D Graphics*) no próprio DOM (*Document Object Model*) do HTML, possibilitando que os navegadores modernos possam renderizar objetos 3D em sua própria árvore de estrutura de marcação sem a necessidade de instalação de *plug-ins*.

A distribuição dos dados referente a renderização da cena 3D é feita através da biblioteca socket.io do node.js. Nesta solução tanto o nó mestre que recebe as interações do usuário, como os nós escravos, trocam informações por intermédio do servidor *Web-Socket*, mantendo assim a estrutura cliente-servidor durante todo o ciclo da execução da aplicação. Isso faz com que toda a aplicação pare de funcionar se houver algum problema de comunicação com este servidor, diferente da abordagem p2p deste trabalho.

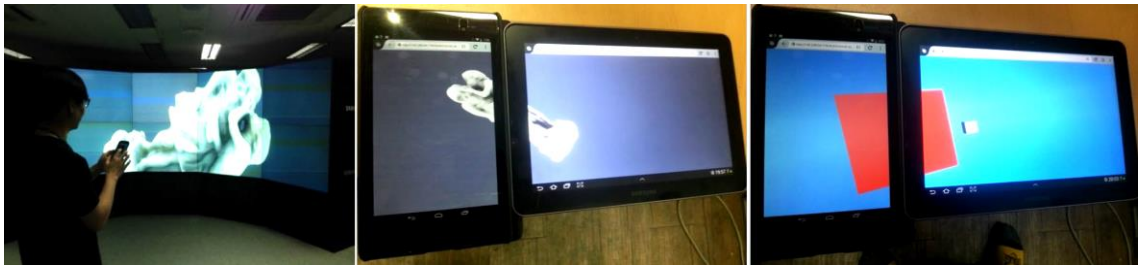

Figura 26 - Cluster com X3DOM (Kim et al., 2015)

Embora essas soluções permitam o desenvolvimento de aplicativos complexos para ambientes virtuais, nenhum deles fornece portabilidade para executar aplicativos em cima de navegadores da *web* sem modificações específicas e, consequentemente, sem a perda do quesito de multi-plataforma.

A arquitetura adotada nesta dissertação é diferente das arquiteturas acima mencionadas pois aborda o problema de desenvolver ambientes de RV totalmente imersivos e interativos usando tecnologias padrão da *web* com a comunicação entre os nós de forma p2p. Como resultado, os desenvolvedores são capazes de tirar proveito das tecnologias que são amplamente utilizadas. Além disso, dado que os navegadores são uma tecnologia multi-plataforma, os aplicativos desenvolvidos para nossa solução ganham em portabilidade.

4.2. Uma arquitetura para aplicações de RV baseadas na *WEB*

Uma aplicação de RV imersiva e interativa em execução em um AG tem como objetivo apresentar múltiplas visões de um mesmo conjunto de dados visuais em tempo real. Essas aplicações tendem a ser desenvolvidas de acordo com a arquitetura Mestre/Escravo por ser a mais adequada para programar e escalar diante de um cenário com baixo número de nós. Apesar de cada processo ser responsável por manter o controle de seu próprio ponto de vista, os processos geralmente têm acesso a todo o conjunto de dados (modelos 3D e texturas) (Cruz-Neira et al., 1992), (DeFanti et al., 2009), (Dias, 2016), (Drolet et al., 2009), (Gnecco et al., 2003).

A sincronização dos dados, que geralmente é implementada com *datalock* e *framelock*, é necessária para suportar a colaboração entre os processos. Neste contexto, *datalock* mantém a coerência dos dados entre processos no *cluster*, garantindo que em

um certo ponto todos os dados relevantes para renderizar são sincronizados em todos os nós. Em aplicações de RV distribuídas, o quadro atual em todas as telas deve corresponder ao que está sendo processado sob o mesmo ponto de vista (e diferentes orientações) e os dados de simulação devem estar em sincronia. *Framelock* (frequentemente referido como *swaplock*) garante que os quadros sejam trocados ao mesmo tempo para todas as telas.

A Figura 27 apresenta uma visão geral da arquitetura desenvolvida nesta dissertação. A camada superior representa as aplicações de RV imersivas e interativas, que podem ser desenvolvidas a partir do zero ou portadas para serem executadas em um ambiente multi-projeção. A abordagem tradicional é executar um aplicativo nativo sobre esta camada. Usando um navegador *web* como um ambiente de execução ganha-se portabilidade, possibilitando que os aplicativos projetados nesta arquitetura possam ser executados em muitas plataformas diferentes (Bergkvist, 2015). Abaixo está a camada de renderização das imagens, que requer um novo quadro a ser renderizado a cada poucas dezenas de milissegundos. Esta atividade é necessária para que sejam renderizadas as cenas coerentes, sem emenda e contíguas. No caso, a WebGL é usada para renderizar imagens 3D no navegador da *web* (Danchilla, 2012). Vale ressaltar que a abordagem comum é usar um motor de jogo (por exemplo, Unity 3D (Jackson, 2015) e Ogre3D (Kerger, 2010) em cima do OpenGL ou DirectX. Há motores equivalentes escritos em JavaScript, como Three.js (Threejs, 2015) e Unity (Jackson, 2015). Na parte superior do sistema operacional está o sistema de comunicação, neste caso WebRTC. WebRTC é implementado em cima do SCTP, o que faz com que seja uma boa solução para o tipo de tráfego de rede. O objetivo desta camada é permitir a comunicação e sincronização entre os processos em execução no *cluster*. Tradicionalmente, uma biblioteca genérica para computação distribuída (por exemplo, *sockets* sobre TCP / UDP, PVM, MPI e libGlass) é usada para atender a este fim, porém estas soluções exigem o desenvolvimento de funcionalidades de alto nível no topo das primitivas de baixo nível disponíveis. A camada inferior é o sistema operacional, que no caso pode ser um *cluster* de computadores composto por nós com diversos sistemas operacionais.

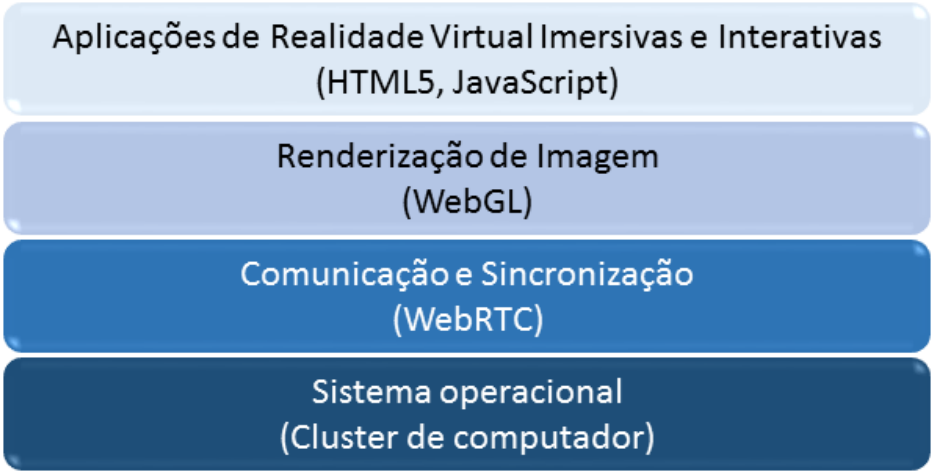

Figura 27 - Arquitetura software de RV imersivo e interativo para ser executado em navegadores

4.2.1. Considerações de desempenho

Tecnologias baseadas em navegadores dependem fortemente de JavaScript, que é uma linguagem de *script* interpretada. Os motores de JavaScript presentes nos navegadores, como o Google V8 ou Mozilla SpiderMonkey, vêm a cada nova versão, recebendo atualizações de melhorias, aumentando cada vez mais seu desempenho no processo de interpretação dos *scripts*. Mas linguagens interpretadas são geralmente mais lentas do que linguagens de programação compiladas como C e C ++. Mesmo com esta desvantagem, os *scripts* são muitas vezes utilizados em desenvolvimentos de jogo, devido a forma dinâmica e a maior facilidade na escrita destes *scripts* (Solihin e Tiwari, 2012).

A forma como os dados são carregados nestas aplicações também é diferente. Enquanto que em uma aplicação compilada e executada de forma nativa no sistema operacional os dados são carregados diretamente do disco rígido, em aplicativos baseados no navegador os dados geralmente vêm de um servidor pela *Internet*. Assim, neste último caso, o processo de inicialização pode ser mais lento. Porém em aplicações muito pesadas, estas podem ser implementadas em forma de pacotes/módulos, reduzindo assim o tempo gasto para a inicialização apenas com os dados relevantes para aquele momento (Solihin e Tiwari, 2012).

Outra diferença também é que não há a necessidade de instalação, bastando apenas a presença de um navegador *web* atualizado e o acesso a aplicação consiste em carregar uma nova URL. Desta forma o custo de implantação e manutenção de aplicações compiladas e nativas torna-se maior. A aplicação compilada cliente precisa ser instalada e configurada em todos os nós do *cluster*, e cada nó precisa baixar os *patches* individualmente quando as atualizações estão disponíveis. Este processo geralmente é feito com ferramentas como Ansible (Hall, 2013) ou Puppet (Hendrix, Benjamin e Yao, 2012), porém adiciona mais uma camada de complexidade que deve ser considerada.

Navegadores atuais, com seus interpretadores JavaScript e implementação WebGL, têm desempenhos muito razoáveis que permitem aplicações gráficas complexas.

4.3. Considerações para sincronização em tempo real de rede

Aplicações de RV imersivas e interativas construídas no topo da arquitetura criada podem ser executadas em qualquer navegador que suporte HTML5, JavaScript, WebGL e WebRTC. Firefox, Chrome, Safari e Opera são exemplos de navegadores que são amplamente utilizados e que suportam essas tecnologias. Uma medida importante a considerar nesta classe de aplicações é a taxa de quadros, que é basicamente o número de quadros processados por segundo (fps), pois quanto maior for esta taxa, maior será a sensação de imersão, a interatividade, e envolvimento. Portanto, isto é importante para avaliar o desempenho da taxa de quadro que pode ser obtida na utilização de tecnologias *web*.

O desempenho da taxa de quadros, no entanto, pode ser afetado por vários fatores. A principal razão para uma taxa de quadro lenta é uma cena complexa que está além da capacidade de processamento da placa de vídeo. Usando WebGL em um navegador é adicionada uma nova camada para o projeto, assim como ocorre em uma linguagem interpretada em vez de código nativo. Ao executar um sistema de renderização distribuída como o proposto aqui, o desempenho da rede é uma questão importante também.

Nesta seção é analisado como as redes podem afetar a comunicação em tempo real entre os nós e, portanto, influenciar o seu desempenho. Na tradicional abordagem de sincronização este é um problema sério, uma vez que as sincronizações são realizadas através da rede e qualquer problema ou atraso que ocorra se reflete diretamente em uma perda de taxa de quadros. Assim, é necessário assegurar que a transmissão de dados através do WebRTC custe pouco tempo de processamento e pouco atraso para alcançar os outros nós.

Também é necessário garantir que esta seja uma solução robusta, capaz de lidar com problemas de rede. As questões mais importantes enfrentadas pelos aplicativos baseados em rede são: (i) a latência, (ii) a variação de tempo na entrega de pacotes, e (iii) a perda de pacotes. Estes problemas tendem a ser encontrados em redes de dados com base na *Internet*, que são usados em soluções de AG geograficamente distribuídas. Sistemas locais que são executados em LANs não são influenciados por esses problemas (Dias, 2016).

Neste trabalho foi utilizada uma ferramenta emuladora de adversidades de rede para testar o desempenho da arquitetura proposta. Esta arquitetura foi avaliada sob diferentes configurações de rede. O interesse em examinar esta proposta é no que diz respeito ao seu desempenho a entrega de pacotes e como ela se adapta a fatores que influenciam negativamente o desempenho da rede.

Foi utilizada NetEm (*Network emulation*), que é uma ferramenta disponível no módulo de controle de tráfego do Linux, que permite que sejam alteradas as configurações do fluxo de pacotes nas interfaces de rede, como atraso de entrega, perda de pacotes, reordenação e variação de tempo de entrega de pacotes e largura de banda. Essa ferramenta faz parte do pacote de ferramentas do Linux iproute2, que é controlado pela ferramenta de linha de comando tc (*traffic control*) (Iqbal, 2013), (Linux, 2016).

As funções dessa ferramenta alteram o fluxo de pacotes, emulando as adversidades configuradas e, consequentemente, atuando diretamente no escalonador de controle de tráfego da interface de rede desejada. O escalonador da ferramenta utiliza um algoritmo do tipo FIFO (*First In Fisrt Out*, primeiro a chegar é o primeiro a sair) para controlar o tráfego (Iqbal, 2013), (Linux, 2016).

Além de suportar configuração via comandos de linha, o NetEm possibilita que sua configuração seja via uma interface gráfica (GUI -*Graphical User Interface*) desenvolvida pela InterWorking Labs, que propõe uma forma mais facilitada para atingir os resultados desejados (InterWorking, 2016).

A Figura 28 mostra a interface gráfica do NetEm, que exibe as propriedades que podem ser configuradas e seus valores correspondentes podem ser editados nos campos disponíveis. Neste exemplo, a propriedade configurada está sendo a largura de banda com o valor de 1,5 Mbps. Assim, essa interface facilita configurações diversas para que sejam executadas durante testes.

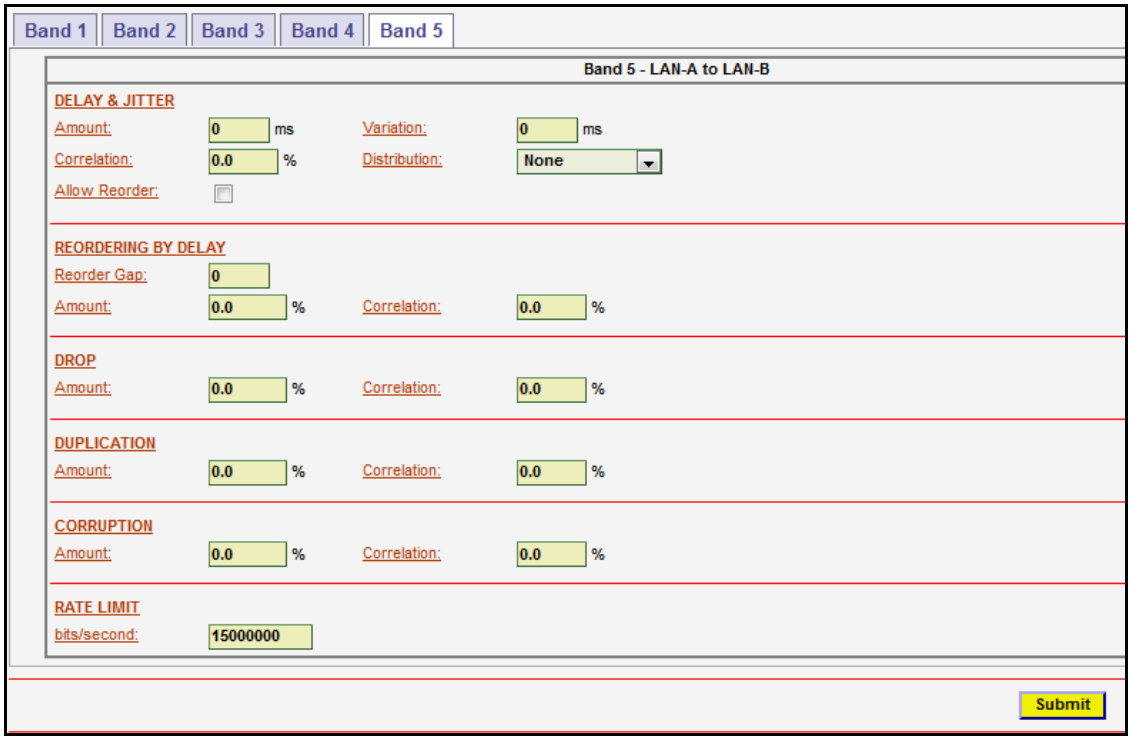

Figura 28 - GUI de configuração do NetEm da InterWorking Labs (InterWorking, 2016)

Neste trabalho o objetivo é utilizar o WebRTC para a realização da comunicação entre os nós. Então, os testes voltaram-se para verificar o desempenho da rede. Para isso, os parâmetros a seguir foram configurados para prejudicar a comunicação na interface de rede dos nós e assim, como resultado, obter uma estimativa do desempenho da solução adotada:

- **Latência**: uma taxa de latência variável foi atribuída a pacotes enviados através de uma interface de rede específica (eth0 / lo). Para realizar este teste foi aplicado o comando *tc qdisc add dev eth0 root netem delay Xms*, onde X é o valor em milissegundos da latência a ser aplicada na placa de rede;

- **Variação no tempo de entrega:** latência em uma rede de dados não é uniforme. Portanto, também deve ser levado em conta simulações de variação de tempo na entrega de pacotes. Foi utilizado o comando *tc qdisc change dev eth0 root netem delay 0ms Xms Y%*, onde X é o valor limite da variação de atraso, que sempre se inicia com 0 neste comando, e Y corresponde a porcentagem probabilística. Neste comando foi adicionado ao trecho correspondente ao *delay* uma variação de atraso, esta variação é chamada de *jitter*, que irá atuar de forma randômica de acordo com a porcentagem configurada. Esta porcentagem indica a chance de que ocorra o atraso nos pacotes que passam pela interface de rede, tornando a simulação mais próxima de uma rede real;

- **Perda de pacotes**: a perda aleatória de pacotes é extremamente prejudicial para sistemas em tempo real, se eles são ou não reenviados pelo emitente, na maioria das vezes, ele irá resultar em inconsistências no ambiente. O comando aplicado para este teste foi *tc qdisc change dev eth0 root netem loss X%*, onde X é o valor em porcentagem da probabilidade de perda independente para os pacotes de saída a partir da interface de rede escolhida;

- **Largura de banda**: larguras de banda diferentes foram utilizadas para avaliar o fluxo de dados com o comando *tc qdisc change dev eth0 root netem rate 256kbps*. Neste exemplo de comando, a rede por onde os pacotes serão enviados terá um limite de 256kbps na sua transmissão.

A escolha do WebRTC sobre *WebSockets* (sessão 3.6.2 Servidores) ao invés de outras tecnologias foi feita por seu protocolo subjacente. WebRTC usa SCTP, protocolo mais apropriado para situações de perda de pacotes e transmissão em tempo real. *Web*

sockets usam TCP, o qual tem um comportamento muito ruim em relação a perda de pacotes, retransmitindo todos os pacotes. Experiências atuais tem resultados bem-sucedidos usando *WebSockets* em LANs (Dias, 2016). Porém neste trabalho, a intenção de apresentar uma aplicação de RV de multi-projeção suportada por navegadores, que podem estar numa mesma rede ou não (quando a aplicação está hospedada na *internet*), resultou na escolha do WebRTC.

O limite mínimo para uma experiência de usuário satisfatória é de 60 quadros por segundo, qualquer taxa de quadros inferior é considerada prejudicial para a experiência do usuário neste trabalho. Assim, os testes realizados foram feitos visando uma emulação de ambiente virtual, a fim de manter a sua consistência, fornecendo pelo menos uma taxa de 60 mensagens por segundo usando WebRTC, considerando que cada mensagem equivale as um grupo de informações enviadas para o ambiente virtual gerar uma atualização de quadro. Os testes foram realizados na estrutura exibida na Figura 29 e descrito nos passos seguintes:

- A rede de dados foi simulada a fim de realizar os testes em diferentes configurações de rede;
- A conexão WebRTC é garantida após uma negociação com um servidor de sinalização implementado em Node.js;
- Dois computadores foram usados para os testes, um para enviar as mensagens (Máquina 1), e um para receber as mensagens (Máquina 2). Eles eram máquinas virtuais sob o mesmo *host* físico;
- Foram enviadas em sequência 200 mensagens de 100 *bytes*. Levando em média 147ms (tempo entre as primeiras mensagens e as últimas mensagens que foram enviadas, e medidas pelo navegador). Nas medições do receptor (Maquina 2), contamos pacotes recebidos dentro de 1 segundo a partir do primeiro que recebeu a mensagem. Isso é suficiente para verificar que as mensagens se comportaram sob um alto rendimento e é muito maior do que o esperado em aplicações RV que são de 60 a 120 fps;

- Nas medições realizadas foram consideradas as médias dos valores obtidos.

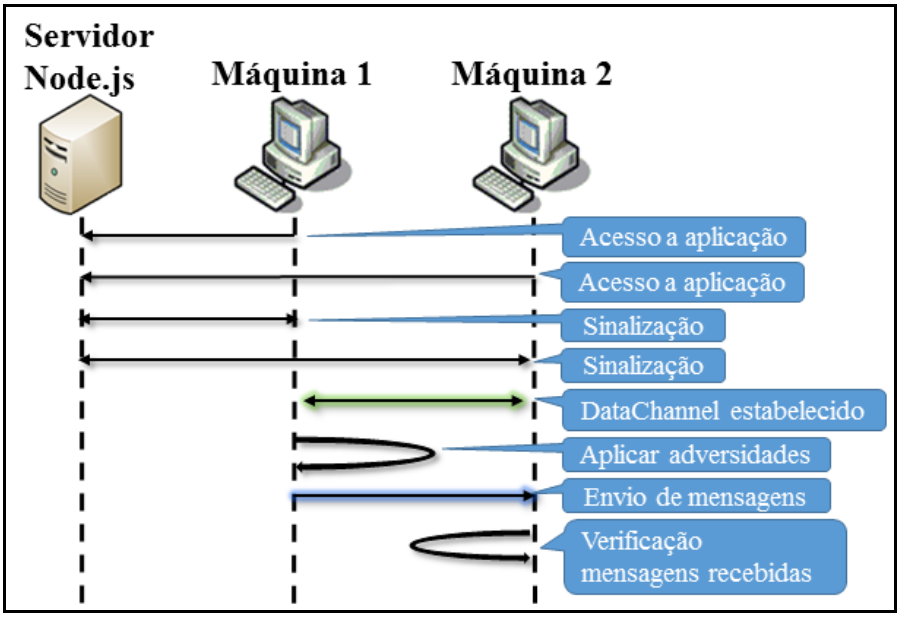

Figura 29 - Teste de envio de mensagens com WebRTC

Os parâmetros das adversidades de rede emulados são mostrados na Tabela 3.

Tabela 3 - Dados das adversidades de rede

Adversidade	**Valores aplicados**
Latência	0ms, 10ms, 100ms, 200ms, 300ms, 400ms, 500ms, 600ms, 800ms e 1000ms
Variação no tempo de entrega	0ms, 5ms, 10ms, 30ms, 50ms, 100ms e 200ms
Perda de pacotes	0,1%, 0,5%, 1%, 2%, 5%, 10% e 25%
Largura de banda	256kb, 1024kb, 5120kb, 10240kb e 102400kb

Embora as redes LAN tenham um bom desempenho, também foi verificado o comportamento de redes menos estáveis, que é importante ser avaliado considerando se a aplicação está interligada em ambientes RV distantes.

A Figura 30 mostra os resultados em relação à perda de pacotes, o que pode ter um efeito altamente prejudicial, uma vez que o comportamento de ambientes de redes de transmissões confiáveis, cada pacote perdido deve ser reenviado. Quando a perda de pacotes é aumentada, a taxa de recepção de mensagens diminui com uma perda de 25%. Considerando que em ambientes reais raramente têm-se uma perda de pacotes tão alta, é

possível observar que sob redes razoáveis a taxa de transferência é mantida como esperado.

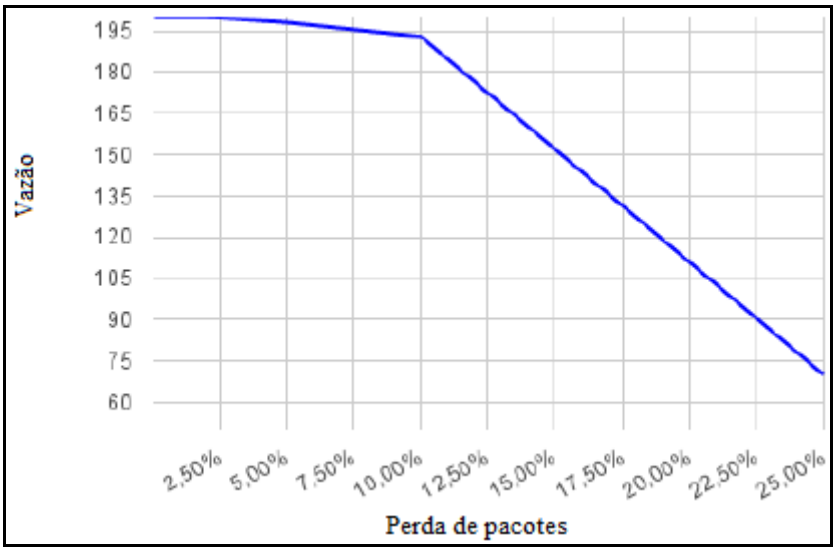

Figura 30 - Perda de pacotes por transferência

A Figura 31 mostra os resultados somente em relação ao tempo de atraso de entrega (*jitter*). Esta variação continua a mudar a taxa de entrega de mensagens, afetando diretamente a taxa de quadros do aplicativo. De acordo com a figura, pode-se dizer que, por volta de 50ms de atraso a entrega começa a cair. No entanto, variações até este valor não são prejudiciais para o ambiente colaborativo, levado em conta que os menores valores de entrega de mensagens ficaram entre 110 e 120 mensagens.

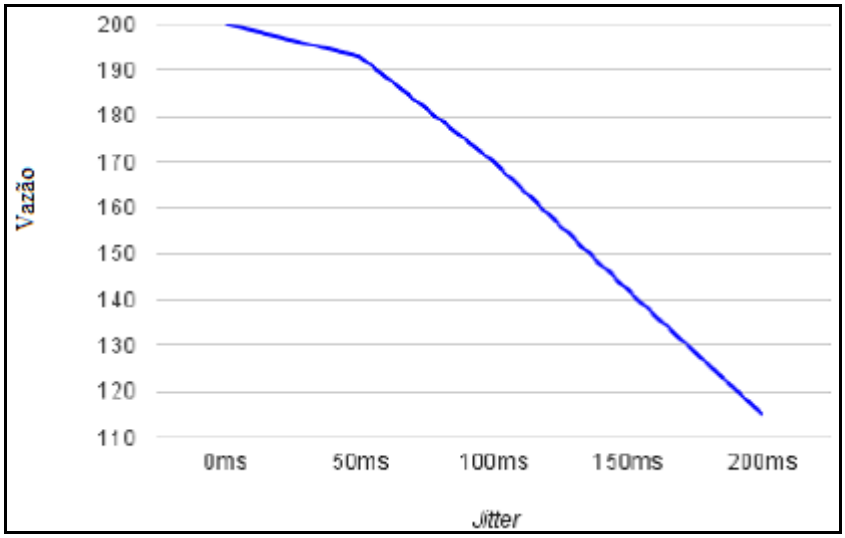

Figura 31 - Tempo de atraso de entrega (*jitter*) por transferência

A Figura 32 mostra os resultados em relação à influência da largura de banda da aplicação. As mensagens trocadas entre as aplicações têm um tamanho médio de 100 *bytes* (apenas atualizações de estado), de modo que a largura de banda utilizada nos testes não teve influência nos resultados, uma vez que mesmo a configuração mais baixa (256kb) é suficiente para realizar a troca de dados.

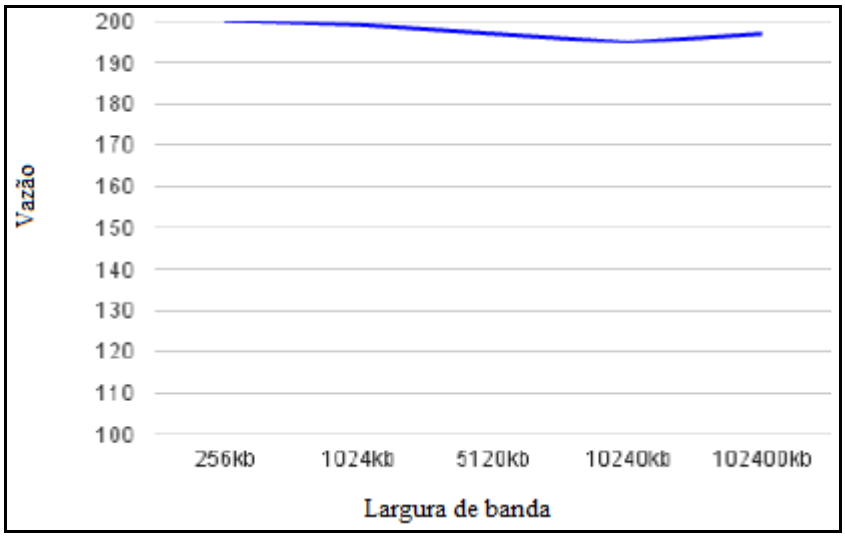

Figura 32 - Largura de banda

Até este ponto, o desempenho da comunicação via WebRTC foi analisado considerando apenas uma dimensão das possíveis adversidades que venham a se manifestar em uma rede com problemas de comunicação. Para simular de forma mais realista, as adversidades aplicadas nos testes anteriores foram combinadas de forma a simular diferentes situações, e com isso analisar o desempenho da solução nestes novos cenários.

A Tabela 4 apresenta a combinação das adversidades do primeiro cenário da rede, porém sem *jitter* aplicado na latência da rede. A Figura 33 mostra o gráfico referente a entrega de pacotes nesse primeiro cenário. Nele é possível verificar que a entrega de mensagens permaneceu em níveis desejados (60 mensagens por segundo) até a aplicação de um valor de latência em 300ms com 3% de perda. Porém, com 5% de perda e valores acima de 200ms a entrega de mensagens ficou abaixo da taxa esperada de mensagens.

Tabela 4 - Adversidades combinadas sem *jitter*

Perda	Larg. Banda	Valores de Latência
1%	256kbps	10ms, 30ms, 60ms, 100ms, 200ms e 300ms
	2048kbps	
3%	256kbps	
	2048kbps	
5%	256kbps	
	2048kbps	

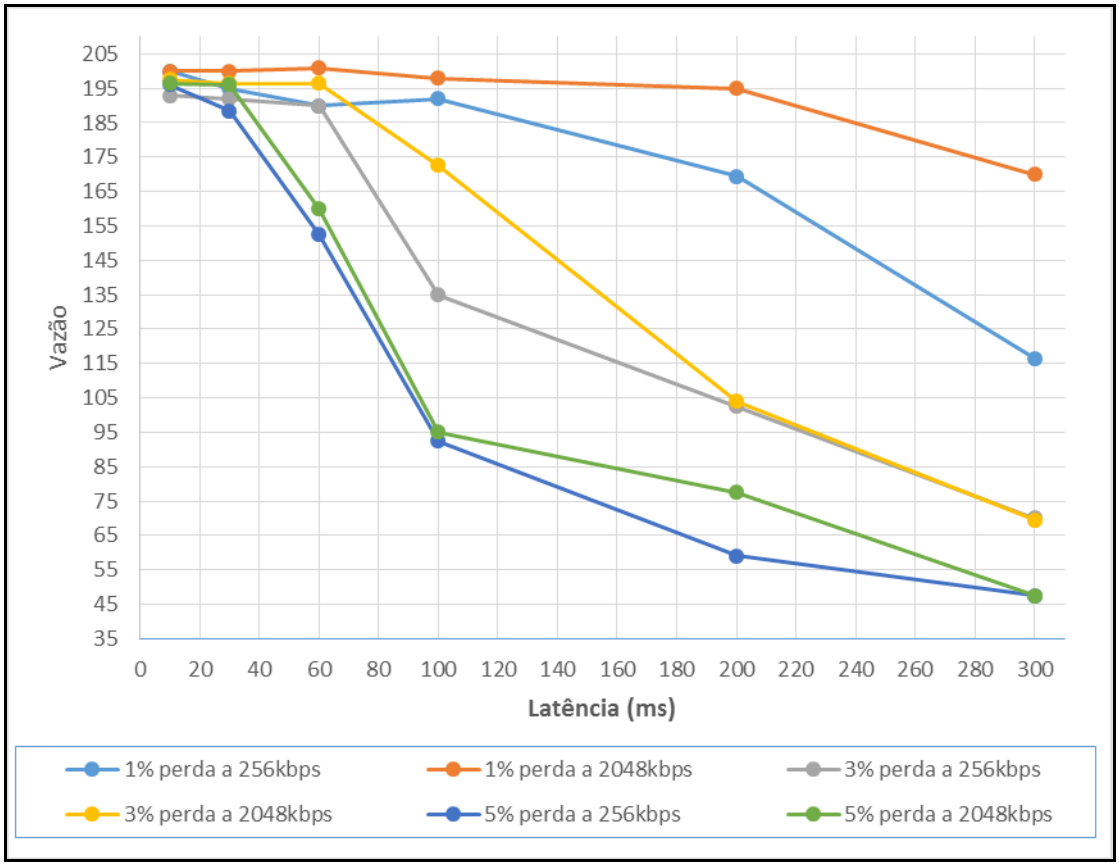

Figura 33 - Resultado da combinação de adversidades sem *jitter*

O segundo cenário com as combinações das adversidades é apresentado pela Tabela 5. A Figura 34 mostra o gráfico resultante, no qual a entrega de mensagens permanece em níveis desejados até a aplicação de 200ms com 5% de *jitter* e 5% de perda de pacotes. Com adversidades acima destes últimos valores, a transmissão de dados afeta negativamente o desempenho da aplicação.

Tabela 5 - Adversidades combinadas com *jitter*

Jitter	Perda	Larg. Banda	Valores de Latência
3ms	1%	256kbps	10ms, 30ms, 60ms, 100ms, 200ms e 300ms
		2048kbps	
	3%	256kbps	
		2048kbps	
5ms	1%	256kbps	
		2048kbps	
	5%	256kbps	
		2048kbps	
10ms	1%	256kbps	
		2048kbps	

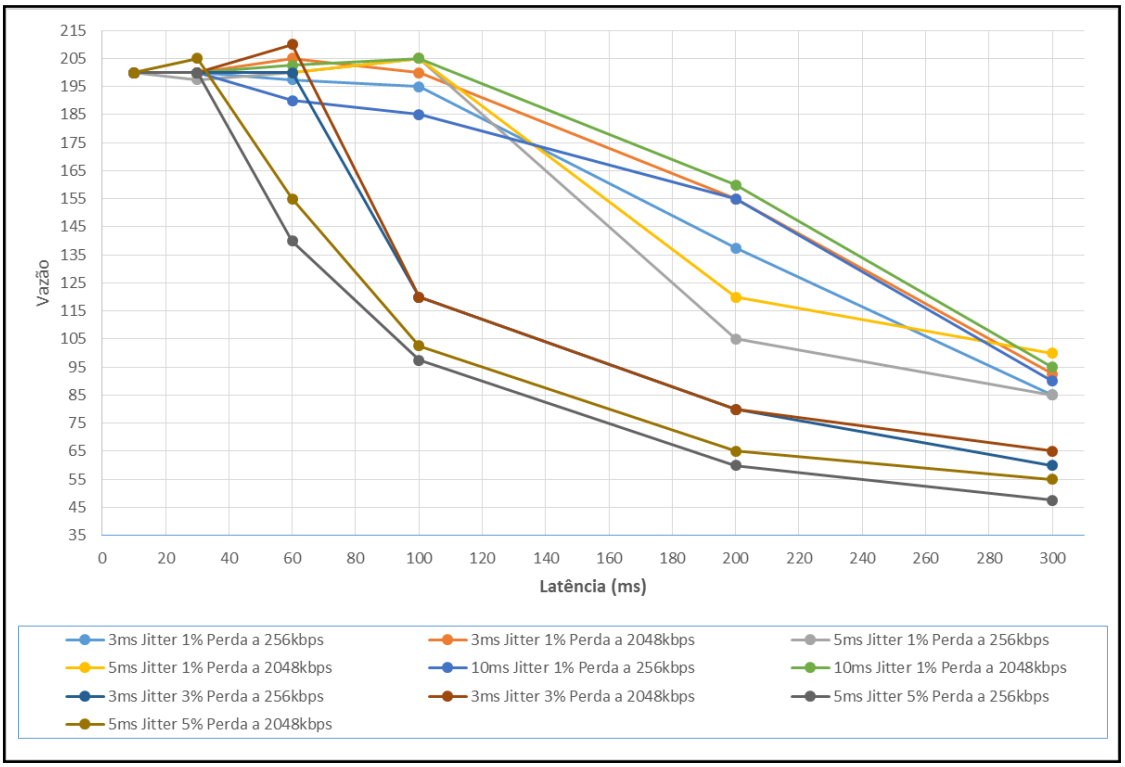

Figura 34 - Resultado da combinação de adversidades com *jitter*

A variação de largura de banda em ambos os cenários foi aplicada visando analisar a influência dela diante da entrega de mensagens afetada pela adversidade. Nos resultados obtidos foi possível observar que diante da variação de largura de banda aplicada não ocasionou diferenças significativas na entrega de pacotes, já que a limitação da

transmissão das mensagens ocorre, de forma mais efetiva, devido a adversidade aplicada.

4.4. Estudo de caso

A fim de mostrar a adequação das tecnologias *web* que apoiam o desenvolvimento de aplicações 3D, foi portada uma aplicação escrita em WebGL VRML/OBJ para um ambiente de mini-CAVE. A Figura 35 mostra o aplicativo portado em execução no ambiente mini-CAVE. Este exemplo mostra que até mesmo um aplicativo relativamente complexo pode ser portado para um ambiente multi-projeção com base em um navegador *web*.

Figura 35 - Visualização do software portado para o ambiente mini-CAVE

A mini-CAVE consiste de um AG com nós (Intel Core i7 8 GB de RAM) rodando Linux (Ubuntu), placas gráficas 3D (NVIDIA FX 1800), e seis projetores de alta definição (BenQ W1000 HD). O ambiente tem três telas com 3D estereoscópico passiva e suporta múltiplos dispositivos de interação. Esta estrutura é ilustrada pela Figura 36.

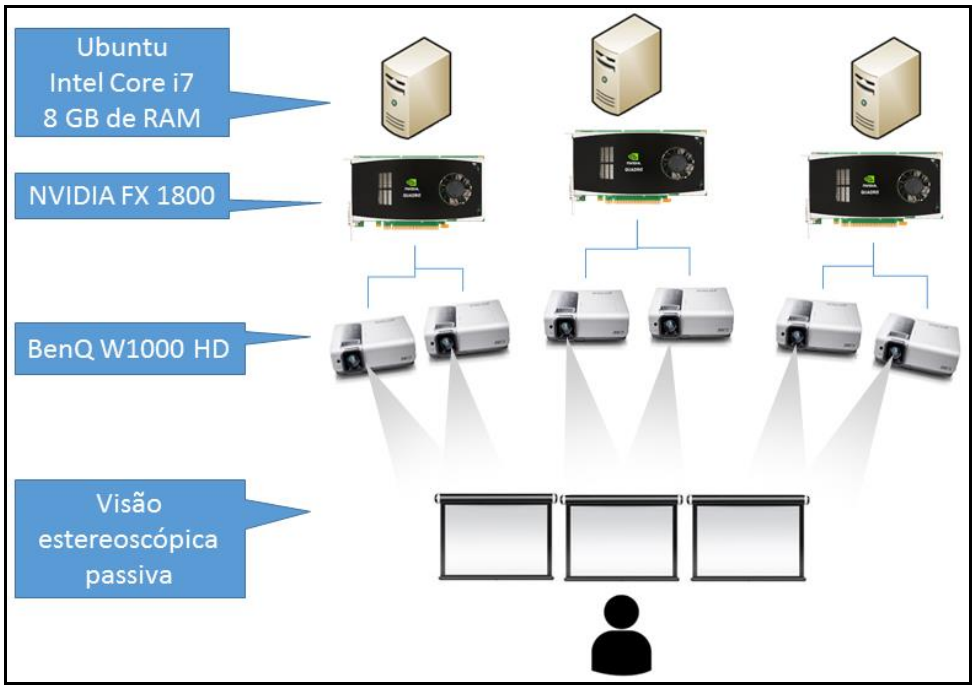

Figura 36 - Estrutura do ambiente mini-CAVE

A interação é uma característica fundamental para o sucesso deste projeto, uma vez que desempenha um papel fundamental no apoio aos usuários ao executar tarefas em RV. A interação pode ser realizada através de dispositivos convencionais, tais como o *mouse* e o teclado, e também por meio de dispositivos não convencionais, tais como luvas de dados e rastreadores de movimentos (Molnar et al., 1994).

A aplicação compartilha dados como posição e orientação do ponto de visão, que é continuamente sincronizado durante a sua execução. O processo de portar a aplicação para o ambiente mini-CAVE implicou em pouca codificação extra, já que tendo a aplicação em WebGL rodando no navegador, a codificação extra consistiu em um novo módulo de sincronização dos dados.

Nesta aplicação foi utilizada a biblioteca *RTCMultiConnection* (sessão 3.6.1.5) para criar o canal de comunicação de dados em WebRTC. Para que o canal seja criado é necessário instanciar o objeto *connection* e a partir deste são feitas as configurações do canal. A Figura 37 mostra um código de configuração de um canal sem suporte a áudio, vídeo (linha 3 e 4), somente habilita a comunicação através de dados de mensagens (linha 5). Em seguida nas linhas 11 a 14 são definidas as informações referentes ao cliente

como ID, ID da sessão e inicializado o processo de sinalização para o canal. Após inicializada a sinalização seguem-se dois processos para a abertura do canal de dados. Caso seja o nó mestre, o canal é aberto juntamente com o comando de criação da sala (linha 17), e se for um nó escravo, o canal é estabelecido com o comando de entrada na sala previamente criada (linha 22).

```
1   var connection = new RTCMultiConnection();
2   connection.session = {
3                           audio : false,
4                           video : false,
5                           data : true
6                       };
7   connection.sdpConstraints.mandatory = {
8                               OfferToReceiveAudio : false,
9                               OfferToReceiveVideo : false
10                          };
11  connection.userid = 'useridTxt';
12  connection.enableFileSharing = false;
13  connection.channel = connection.sessionid = 'sessionid';
14  var signaler    = initReliableSignaler(connection, '/');
15
16  //para o master
17  connection.open(
18          signaler.createNewRoomOnServer(connection.sessionid)
19                  );
20
21  //para os clientes
22  connection.join({
23                      sessionid : sessionid,
24                      extra : {},
25                      session : connection.session
26                  });
```

Figura 37 - Configuração do canal de dados

Com o canal de dados estabelecido é possível a troca de dados através de mensagens, a Figura 38 mostra um código de como são feitos o envio e o recebimento. Nesta parte da aplicação, considerando que o único nó que recebe as interações do usuário é o nó mestre, no método que atualiza os objetos 3D na tela (linha 2), é necessário que seja disparada uma mensagem com os valores atualizados para os clientes (linha 17). Já no código dos clientes, ao receber a mensagem (linha 26), os dados atualizados devem ser encaminhados para o método que atualiza os objetos 3D na tela (linha 28).

```
1   //envio de dados do master
2   this.update = function(){
3       //guarda a posição atual
4       var position = this.object.position;
5       ...
6       //calcular a nova posição
7       offset.copy(position).sub(this.target);
8       ...
9       //aplica a nova posição
10      ...
11      //envia para os clientes
12      if(connection.userid == 'master'){
13          connection.extra = {
14              offset2 : offset,
15              from : connection.userid
16          };
17          connection.send(connection.userid +
18                          ' Updated value: pos('+
19                          offset +
20                          ')'
21                      );
22      }
23  }
24
25  //recebimento de dados nos clientes
26  connection.onmessage = function(event) {
27      if (connection.userid != event.extra.from) {
28          this.update2(event.extra.offset2);
29      }
30  }
31
32  this.update2 = function(offset2) {
33      offset.x = offset2.x;
34      offset.y = offset2.y;
35      offset.z = offset2.z;
36      ...
37      //aplica nova posição
38      position.copy(this.target).add(offset);
39      //mudar a perspectiva deste cliente baseado
40      //no novo offset recebido
41      ...
42  }
```

Figura 38 - Envio recebimento

Para que a aplicação portada trabalhe com barreiras de sincronização (*frame-lock*), tendo assim uma coerência nas imagens exibidas em todos os monitores, é necessário alterar o código (Dias, 2016), (Guimarães, 2004). As Figuras 39 e 40 mostram o código alterado de forma resumida.

```
1   //envio de dados do master
2   this.update = function(){
3       //guarda a posição atual
4       var position = this.object.position;
5       ...
6       //calcular a nova posição
7       offset.copy(position).sub(this.target);
8       ...
9       //envia para os clientes
10      if(connection.userid == 'master'){
11          connection.extra = {
12              offset2 : offset,
13              bufferOK : False,
14              render : False,
15              from : connection.userid
16          };
17          connection.send(connection.userid +
18                          ' Updated value: pos('+
19                          offset +
20                          ')'
21                      );
22      }
23      //enviar para o buffer
24      ...
25  }
26
27  //recebimento de dados no master
28  connection.onmessage = function(event) {
29      if ((connection.userid != event.extra.from) &&
30          (event.extra.bufferOK == True) ){
31          if(arrayClients.indexOf(event.extra.from) == -1){
32              arrayClients.push(event.extra.from);
33          }
34
35          connection.getAllParticipants().forEach(function(participantId) {
36              cont = cont + 1;
37          });
38
39          if( arrayClients.length == (cont-1) ){
40              if(connection.userid == 'master'){
41                  connection.extra = {
42                      offset2 : null,
43                      bufferOK : null,
44                      render : True,
45                      from : connection.userid
46                  };
47                  connection.send(connection.userid + ' Render All');
48              }
49              arrayClients = [];
50          }
51      }
52
53      if (event.extra.render == True){
54          liberaBuffer();
55      }
```

Figura 39 - Framelock estruturado, parte 1

```
56  }
57
58  //recebimento de dados nos clientes
59  connection.onmessage = function(event) {
60      if ((connection.userid != event.extra.from) &&
61          (event.extra.render == False) ){
62          this.update2(event.extra.offset2);
63      }
64      if (event.extra.render == True){
65          liberaBuffer();
66      }
67  }
68
69  this.update2 = function(offset2) {
70      offset.x = offset2.x;
71      offset.y = offset2.y;
72      offset.z = offset2.z;
73      ...
74      //mudar a perspectiva deste cliente baseado
75      //no novo offset recebido
76      ...
77      //enviar para o buffer
78      ...
79      //envia mensagem para o master
80      connection.extra = {
81          offset2 : null,
82          bufferOK : True,
83          render : null,
84          from : connection.userid
85      };
86      connection.send(connection.userid + ' bufferOK ');
87  }
88
89  this.liberaBuffer = function(){
90      //comando para exibir os comandos armazenados em buffer
91      gl.flush();
92  }
```

Figura 40 - Framelock estruturado, parte 2

Desta forma este trabalho apresenta a arquitetura proposta como uma solução promissora, que suporta aplicações semelhantes a está apresentada neste estudo, com pouco esforço.

A aplicação foi executada sem problemas, conseguindo 60 mensagens por segundo na maioria das vezes, com o desempenho de renderização alcançado semelhante ao rendimento em uma rede local mesmo com as configurações de rede afetadas pelas adversidades, exceto nos níveis de adversidades aplicadas com valores acima de 200ms de latência com 5% de *jitter* e perda, como já foi mostrado na Figura 34.

4.5. Considerações finais do capítulo

Os avanços nas tecnologias de *hardware* e *software* permitiram que os navegadores *web* modernos carreguem e renderizem cenas 3D complexas em tempo real. Aproveitando os benefícios fornecidos por essas tecnologias, neste trabalho foi apresentada uma arquitetura para o uso delas como solução de visualização de aplicações imersivas e interativas 3D. Assim, a portabilidade da aplicação é garantida através dos navegadores, uma vez que estão presentes na maioria das plataformas operacionais.

Outra vantagem da arquitetura criada é que os desenvolvedores podem se concentrar no desenvolvimento da camada de aplicação, e tomar proveito dos processos abstraídos, referente à distribuição das informações entre os nós.

Capítulo 5 - Conclusão

As aplicações de RV vem inspirando pesquisas nos meios científicos de várias áreas, e isso vem promovendo a evolução desta tecnologia. Este trabalho concentra-se especificamente em aplicações de RV imersivas e interativas voltadas para a execução em ambientes de multi-projeção.

Para a introdução deste assunto, foi realizada uma pesquisa bibliográfica que apresenta conceitos de RV, sistemas de multi-projeção, *clusters* de computadores e AG. Com o objetivo de contextualização das aplicações de RV no cenário atual das aplicações *web*, este trabalho foi motivado em apresentar a viabilidade da execução destas aplicações em navegadores *web* e a utilização do WebRTC como facilitador da comunicação dos nós de um AG baseado em navegadores.

De forma a viabilizar a execução de tais aplicações em navegadores como plataforma, foi idealizada e apresentada uma arquitetura que visa auxiliar o desenvolvimento.

Devido aos avanços no suporte às tecnologias gráficas dos navegadores, criou-se um novo ambiente de execução de aplicações de RV a ser explorado, e dessa forma, estas aplicações que antes dependiam de instalações locais de *softwares* e *drives* de *hardware*, agora ganham em portabilidade e agilidade na distribuição para os clientes.

Porém, os sistemas de multi-projeção, quando estabelecidos em ambientes *web*, necessitam de uma solução, que de forma compatível com a portabilidade da parte gráfica das aplicações, também seja para a distribuição de informações entre os nós de um AG.

Diante destas perspectivas, este trabalho apresentou, também através de pesquisas bibliográficas, conceitos da comunicação p2p com WebRTC e trabalhos desenvolvidos pela comunidade cientifica que apoiam e confirmam o potencial dos estudos nesta ferramenta, e que também faz parte da arquitetura proposta neste trabalho.

Devido p2p ser a forma pela qual a troca de informações é realizada na arquitetura proposta para aplicações de multi-projeção em navegadores *web*, esta forma de

comunicação necessita garantir que a entrega acontecerá e que não causará atrasos na renderização do ambiente virtual. Então, baseando-se no padrão mínimo de 60 quadros por segundo que uma aplicação de RV estereoscópica necessita proporcionar, para que assim mantenha características de imersão e interação com o usuário, foram realizados testes de envio e recebimento de mensagens em uma rede de dados com as características de latência, perda de pacotes e largura de banda afetadas de forma negativa. Desta forma foi possível avaliar o desempenho do WebRTC nesta tarefa de distribuição de informações em um AG.

A princípio foram realizados testes individuais em cada característica de rede, onde os valores das adversidades eram aumentados gradativamente. Nestes testes iniciais foi verificado que a entrega de mensagens satisfez o nível mínimo em todos eles. Porém, com o objetivo de simular uma situação mais realista de uma rede com problemas de comunicação, foram criados dois cenários em que a rede fosse afetada por mais de uma adversidade ao mesmo tempo. Foi observado que o comportamento permaneceu praticamente inalterado diante de combinações de adversidades com baixo valor de interferência, demonstrando um bom desempenho mesmo diante das adversidades combinadas e com os valores de interferência progressivos.

No primeiro cenário foram combinadas as adversidades de perda de pacotes, largura de banda e latência de rede sem *jitter*. Já no segundo cenário foram combinadas a mesmas adversidades do cenário anterior, porém adicionada a variação de *jitter*. Em ambos, foi observado que a solução não manteve o nível de entrega de mensagens quando aplicadas latências acima de 200ms combinadas com 5% de perda de pacotes. Este baixo desempenho com latências acima de 200ms também é observado por outros autores que realizaram testes semelhantes ao deste trabalho como Chen (2005) e Sung et al. (2006).

Já a largura de banda sofrendo alterações entre 256kbps e 2048kbps não apresentou grandes alterações na entrega dos pacotes mesmo quando aplicadas adversidades combinadas em valores maiores, pois independentemente do tamanho da largura de banda, a entrega é afetada de forma mais presente pelas adversidades de latência e perda

de pacotes, já que as mensagens de atualizações que a aplicação transmite tem tamanho reduzido.

Como trabalho futuro é possível avaliar a adequação desta arquitetura para aplicações mais complexas e estudar a sincronia dos sistemas geograficamente distantes em detalhes. Além disso, sugere-se o desenvolvimento de uma API para atenuar os esforços de desenvolvimento e aporte de aplicações para arquiteturas subjacentes.

REFERÊNCIAS

Abbasi, A and Baroudi, U (2012). “Immersive environment: An emerging future of telecommunications”. IEEE MultiMedia, Volume: 19, Issue: 1, pages 80-80.

Amaral, V. M. F. (2013). “Framework e Cliente WebRTC”, 126 f. Dissertação de Mestrado, Universidade do Minho, Braga, Portugal.

Bues, M; Blach, R; Stegmaier, S; Häfner, U; Homann, H. and Haselberger, F. (2001). “Towards a scalable high performance application platform for immersive virtual environments”. In Proceedings of the 7th Eurographics Conference on Virtual Environments; 5th Immersive Projection Technology, EGVE'01, pages 165-174, Aire-la-Ville, Switzerland, Switzerland, 2001. Eurographics Association.

Bergkvist, A. (2015) “Webrtc 1.0: Real-time communication between browsers:w3c editor’s draft”. Disponível em: http://w3c.github.io/webrtc-pc/ Acessado em janeiro 2016.

Chen, L. (2005). “Effects of network characteristics on task performance in a desktop cve system”. In Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on (Vol. 1, pp. 821-826). IEEE.

Costa, D. G. (2007). “Comunicações Multimídia na Internet”. 1 Ed. Rio de Janeiro, Ciência Moderna. 256 p.

Cruz-Neira, C.; Sandin, D.J.; DeFanti, T.A.; Kenyon, R.V.; Hart, J.C. (1992). “The cave: audio visual experience automatic virtual environment”. Commun. ACM 35, 64–72. DOI http://doi.acm.org/10.1145/129888.129892. URL http://doi.acm.org/10.1145/129888.129892.

Danchilla, B. (2012). “Beginning WebGL for HTML5”. Apress, Berkely, CA, USA, 1st edition, 2012.

Davis, D. e Nyman, R. (2013). “Cross-browser camera capture with getUserMedia/WebRTC”. Disponível em: <https://hacks.mozilla.org/2013/02/cross-browser-camera-capture-with-getusermediawebrtc/> Acessado em janeiro 2016.

DeFanti, T.A.; Leigh, J.; Renambot, L.; Jeong, B.; Verlo, A.; Long, L.; Brown, M.; Sandin, D.J.; Vishwanath, V.; Liu, Q.; Katz, M.J.; Papadopoulos, P.; Keefe, J.P.; Hidley, G.R.; Dawe, G.L.; Kaufman, I.; Glogowski, B.; Doerr, K.U.; Singh, R.; Girado, J.; Schulze, J.P.; Kuester, F.; Smarr, L. (2009). "The optiportal, a scalable visualization, storage, and computing interface device for the optiputer". Future Generation Computer Systems 25(2), 114–123. DOI 10.1016/j.future.2008.06.016. URL http://www.sciencedirect.com/science/article/pii/S0167739X08001040.

Dias, D. R. C. (2011). "Sistema Avançado de Realidade Virtual para Visualização de Estruturas Odontológicas". Dissertação (Mestrado) — IBILCE/UNESP, São José do Rio Preto, 2011.

Dias, D. R. C. (2016). "Uma arquitetura para intercomunicação de ambientes de realidade virtual distribuídos baseados em aglomerados gráficos remotos", 170 f. Tese de Doutorado, Universidade Federal de São Carlos, São Carlos, São Paulo.

Dias, D. C.; La Marca, A.; Moia Vieira, A.; Neto, M.; Brega, J.; Guimarães, M. P; Lauris, J. (2010). "Dental arches multi-projection system with semantic descriptions". In: Virtual Systems and Multimedia (VSMM). 16th International Conference on Virtual Systems and Multimedia. Seoul: IEEE Xplore, 2010. (VSMM 2010, ISBN 978-1-4244-9027-1), p. 314 –317.

Drolet, F.; Mokhtari, M.; Bernier, F.; Laurendeau, D. (2009). "A software architecture for sharing distributed virtual worlds". In: Virtual Reality Conference, 2009. VR 2009. IEEE, pp. 271–272. DOI 10.1109/VR.2009.4811050.

Eclipse Foundation (2016). "Vert.X". Disponível em: < http://vertx.io/ >. Acessado em novembro 2016.

Finkelstein, S and Suma, E. A. (2011). "Astrojumper: Motivating exercise with an immersive virtual reality exergame". Presence: Teleoper. Virtual Environ., Feb. 2011.

Froehlich, B. e Livingston, M. (2014). "Quo Vadis CAVE: Does Immersive Visualization Still Matter?". IEEE Computer Graphics and Applications, New Brunswick, Nova Jersey, EUA, p. 16-17, 2014, Sep./Oct. 2014.

Gnecco, B.B.; Guimarães, M.P.; Zuffo, M.K. (2003). “Um framework para computação distribuída”. In: Simpósio Brasileiro de Realidade Virtual. SBC, Ribeirão Preto 2003.

Grigorik, I. (2013). High-Performance Browser Networking. O’REILLY, Sebastopol, CA, First Edition 2013.

Guimarães, M. P. (2004). “Um ambiente para o desenvolvimento de aplicações de realidade virtual baseadas em aglomerados gráficos”, 126 f. Tese de Doutorado, Escola Politécnica da Universidade de São Paulo, São Paulo, 2004.

Guimarães, M. et al. (2003). “Programação Distribuída Aplicada à Realidade Virtual”, Simpósio Brasileiro de Realidade Virtual, Capítulo 1, Ribeirão Preto, São Paulo.

Hall, D. (2013). Ansible Configuration Management, 92 f. Packt Publishing.

Hasso-Plattner-Institut (2017). “Tele-Board HPI”. Disponível em: < https://tele-board.de> Acessado em janeiro 2017.

Hendrix, V.; Benjamin, D.; Yao, Y. (2012). “Scientific cluster deployment and recovery using puppet to simplify cluster management”. Journal of Physics: Conference Series 396(4), 042,027. Disponível em: http://stacks.iop.org/1742-6596/396/i=4/a=042027.

Hoang, T. N.; Reinoso M.; Vetere, F. and Tanin E. (2016). “Onebody: Remote Posture Guidance System using First Person View in Virtual Environment”. In NordiCHI '16 Proceedings of the 9th Nordic Conference on Human-Computer Interaction, article No. 25, October 2016.

Humphreys, G; Eldridge, M; Buck, I; Stoll, G; Everett, M. and Hanrahan, P. (2001). “Wiregl: A scalable graphics system for cluster”s. In Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '01, pages 129-140, New York, NY, USA, 2001. ACM.

Humphreys, G; Houston, M; Ng, R; Frank, R.; Ahern, S; Kirchner, P. D. and Klosowski, J. T. (2002). “Chromium: A stream-processing framework for interactive rendering on clusters”. ACM Trans. Graph., pages 693-702, July 2002.

Ignácio A. A. V. e Filho V. J. M. F. (2002). “MPI: Uma Ferramenta Para Implementação Paralela. Pesquisa Operacional”, vol.22 no.1, Rio de Janeiro Jan./June 2002. Disponível em: <http://dx.doi.org/10.1590/S0101-74382002000100007>. Acessado em outubro 2016.

InterWorking Labs (2016). "Netem with a GUI". Disponível em <http://iwl.com/maxwell-family/netem-with-a-gui>. Acessado em novembro 2016.

Iqbal F. (2013). “Performance Evaluation of Robust Header Compression Protocol for Low Data Rate Networks”. 91 p. Master thesis. University of Agder. Grimstad, Norway, 2013.

Isakovic, K.; Dudziak, T.; K¨ochy, K. (2002). “X-rooms”. In: Proceedings of the Seventh International Conference on 3D Web Technology, Web3D ’02, pp. 173– 177. ACM, New York, NY, USA. DOI 10.1145/504502.504530. URL http://doi.acm.org/10.1145/504502.504530.

Jackson, S. (2015). “Unity 3D UI Essentials”. Packt Publishing, 2015.

Jamsa, K. (2013). “Introduction To Web Development Using HTML 5”. Jones and Bartlett Publishers, Inc., USA, 2013.

Kemeny, A. (2014). “From driving simulation to virtual reality”. In Proceedings of the 2014 Virtual Reality International Conference, VRIC '14, page 32, New York, NY, USA, 2014. ACM.

Kerger, F. (2010). “OGRE 3D 1.7 Beginner's Guide”. Packt Publishing, 2010.

Khan, M. (2016). "WebRTC Experiments". Disponível em: <https://www.webrtc-experiment.com/>. Acessado em fevereiro 2016.

Kim, H. et al. (2015). “Cluster rendering on large high-resolution multi-displays using X3DOM and HTML”, Springer-Verlag Berlin Heidelberg, 19 nov. 2015.

Kirner, C. (1997). “Introdução a Realidade Virtual”, 1º Workshop de Realidade Virtual, Universidade Federal de São Carlos, São Carlos, São Paulo.

Koskela, T. et al. (2015). “RADE: Resource-aware Distributed Browser-tobrowser 3D Graphics Delivery in the Web”. In: IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob). Center for In-

ternet Excellence, University of Oulu, Oulu, Finland. pp. 500-508. 978-1-4673-7701-0/15. (2015).

Kuntz, S. (2015). “Middlevr a generic vr toolbox”. In: 2015 IEEE Virtual Reality (VR), pp. 391–392. DOI 10.1109/VR.2015.7223460.

Lachapelle, S. (2013). “What was the history of WebRTC inside Google Before it was released to the public?”. Disponível em: <https://www.quora.com/What-was-the-history-of-WebRTC-inside-Google-before-it-was-released-to-the-public?ref=fb> acessado em fevereiro 2016.

Levent-Levi. T. (2014). “How to Implement Sdreen Sharing in WebRTC”. Disponível em: <https://bloggeek.me/implement-screen-sharing-webrtc/> acessado em janeiro 2015.

Li, H.; Kulik, L. and Ramamohanarao, K. (2016). “Automatic Generation and Validation of Road Maps from GPS Trajectory Data Sets”. In CIKM '16 Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, pages 1523-1532, ACM New York, NY, USA 2016

Li, M. and Buchthal, S. (2012). “Advisory services in the virtual world: An empowerment perspective”. Electronic Commerce Research,Volume 12, Issue 1, pp 53–96.

Linux (2016). “Linux Foundation Wiki project collaboration site”. Disponível em:< https://wiki.linuxfoundation.org/networking/netem#dokuwiki__top> Acessado em outubro 2016.

Lochtefeld, M.; Kruger, A. and Gellersen, H. (2016). “DeceptiBike - Assessing the Perception of Speed Deception in a Virtual Reality Training Bike System”. In NordiCHI '16 Proceedings of the 9th Nordic Conference on Human-Computer Interaction, article No. 40, October 2016.

Loreto, S. and Romano, S. P (2015). “Real-Time Communication with WebRTC”. 1 Ed. O’Relly books Media. 164 p. Gravenstein Highway Norty, Sebastopol, Canada.

Lozano, A. A. (2013). “Performance analysis of topologies for Web-basead Real-Time Communication (WebRTC)”, Aalto University School of Electrical Engeneering, Aauto, Finland.

MiddleVR (2016). “MiddleVR: Improve Reality!”. Disponível em: < http://www.middlevr.com/ >. Acessado em novembro 2016.

Molnar, S.; Cox, M.; Ellsworth, D.; Fuchs, H. (1994). “A sorting classification of parallel rendering”. IEEE Comput. Graph. Appl. 14(4), 23–32. DOI 10.1109/38.291528. Disponível em: http://dx.doi.org/10.1109/38.291528.

Moreau, G. (2013). “Visual immersion issues in virtual reality: A survey”. In Proceedings of the 2013 26th Conference on Graphics, Patterns and Images Tutorials, SIBGRAPI-T '13, pages 6-14, Washington, DC, USA, 2013. IEEE Computer Society.

MPI (2016). "MPI Documents". Disponível em: http://mpi-forum.org/docs/. Acessado em outubro 2016.

Muhanna, M. A. (2015). “Virtual reality and the cave”. J. King Saud Univ. Comput. Inf. Sci., July 2015.

Neto, M.P.; Dias, D.R.C.; Trevelin, L.C.; Guimarães, M.P.; Brega, J.R.F. (2015) “Unity Cluster Package – Dragging and Dropping Components for Multi-projection Virtual Reality Applications Based on PC Clusters”, pp. 261–272. Springer International Publishing, Cham. DOI 10.1007/978-3-319-21413-9 19

Nguyen, D. et al. (2016). “Real- Time Optimized NFV Architecture for Internetworking WebRTC and IMS”. Ecole de technologie Superieure, University of Quebec Montreal, Quebec, Canada, H3C lK3. pp. 81-88. 978-1-4673-8991-4/16. (2016).

Nielsen, F. (2016). “Introduction to HPC with MPI for Data Science”. 1 Ed. Springer International Publishing. 282 p. Springer International Publishing Switzerland.

Node.js Foundation (2016). “Node.js”. Disponível em <https://nodejs.org/en/>. Acessado em novembro de 2016.

Pedras, B. F. V; Raposo, A. and Santos, I. H. F. (2013). “Apprc: A framework for integrating mobile communication to virtual reality applications”. In Proceedings of the 12th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry, VRCAI'13, pages 305-308, New York, NY, USA, 2013. ACM.

PeerJS (2016). “The PeerJS library”. Disponível em < http://peerjs.com/>. Acessado em novembro 2016.

Pierleoni, P. et al. (2016). “An Innovative WebRTC Solution for e-Health Services”. In: IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom). Department of Information Engineering Università Politecnica delle Marche, Ancona, 60131, Italy. (2016).

Pinho, M. S. (2000). ”Interação em Ambientes Tridimensionais”. 3rd Brasilian Workshop on Virtual Relity. Gramado, Rio Grande do Sul, Brazil 2000.

Priologic Software Inc (2016). "Priologic's EasyRTC won Best WebRTC Tool Award at WebRTC Conference & Expo". Disponível em < https://www.easyrtc.com/blog/2012-11-30-priologics-easyrtc-won-best-webrtc-tool-award-at-webrtc-conference-expo/>. Acessado em novembro 2016.

Ranasinghe, N.; Do, E. (2016). “Digital Lollipop: Studying Electrical Stimulation on the Human Tongue to Simulate Taste Sensations”. Journal ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), Volume 13 Issue 1, November 2016.

Rtc.io (2016). “rtv.io OpenSource WebRTC”. Disponível em: http://rtc.io/index.html Acessado em janeiro 2016.

Soares, L. P; Raffin, B; Jorge. J. A. (2008). “PC Clusters for Virtual Reality”, The International Jounal of Virtual Reality, p. 67-80, 1 jul. 2008.

Soares, L.P.; Zuffo, M.K. (2004). “Jinx: An x3d browser for vr immersive simulation based on clusters of commodity computers”. In: Proceedings of the Ninth International Conference on 3D Web Technology, Web3D ’04, pp. 79–86. ACM, New York, NY, USA. DOI 10.1145/985040.985052. Disponível em: http://doi.acm.org/10.1145/985040.985052.

Solihin, Y.; Tiwari, D. (2012). “Architectural characterization and similarity analysis of sunspider and google’s v8 javascript benchmarks”. In: Proceedings of the 2012 IEEE International Symposium on Performance Analysis of Systems & Software, ISPASS’12, pp. 221–232. IEEE Computer Society, Washington, DC, USA. DOI

10.1109/ISPASS.2012.6189228. Disponível em: http://dx.doi.org/10.1109/ISPASS.2012.6189228.

Sung, M. Y., Yoo, Y., Jun, K., Kim, N. J., & Chae, J. (2006). “Experiments for a collaborative haptic virtual reality”. In Artificial Reality and Telexistence--Workshops, 2006. ICAT'06. 16th International Conference on (pp. 174-179). IEEE.

TechEmpower (2016). “Web Framework Benchmarks”. Disponível em: <https://www.techempower.com/benchmarks/#section=data-r8&hw=i7&test=plaintext> Acessado em janeiro, 2016.

Temasys Communications Pte Ltda (2016). “Plugin: The power and freedom of WebRTC now avaliable for Intyernet Explorer end Safari”, Disponível em: <http://temasys.com.sg/plugin/#free-plugin> Acessado em janeiro, 2016.

Threejs (2015). “Javascript 3d library”. Disponível em: http://threejs.org/ Acessado em janeiro 2016.

Zuffo, J. et al. (2001). “Caverna Digital – Sistema de Multiprojeção Estereoscópico Baseado em Aglomerados de PCs para Aplicações Imersivas em realidade Virtual”, Escola Politécnica da USP, São Paulo, São Paulo.

WebRTC (2016). “Supported Browsers & Platforms”. Disponível em: <https://webrtc.org/> Acessado em março 2016.

WebRTC, G. (2016). “RTCWeb”. Disponível em: https://webrtcglossary.com/rtcweb/. Acessado em novembro 2015.

Wenzel, M. Meinel, C. (2016). “Full-Body WebRTC Video Conferencing in a Web-Based Real-Time Collaboration System”. In: IEEE 20th International Conference on Computer Supported Cooperative Work in Desing.Hasso Plattner Institute Potsdam Prof. Dr. Helmert Str. 2-3, Potsdam, Germany. pp. 334-339. 978-1-5090-1915-1/16. (2016).