MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

LUCIANO DE SOUZA BARREIRA
MATHEUS BORTOLINI DE OLIVEIRA

FERRAMENTA DE STREAMING DE JOGOS DE FUTEBOL DE ROBÔS DA CATEGORIA ROBOCUP SMALL SIZE LEAGUE

Rio de Janeiro
2019

INSTITUTO MILITAR DE ENGENHARIA

LUCIANO DE SOUZA BARREIRA
MATHEUS BORTOLINI DE OLIVEIRA

FERRAMENTA DE STREAMING DE JOGOS DE FUTEBOL DE ROBÔS DA CATEGORIA ROBOCUP SMALL SIZE LEAGUE

Projeto de Fim de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Engenheiro de Computação.

Orientador: Prof. Paulo Fernando Ferreira Rosa - Ph.D.

Rio de Janeiro
2019

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80 - Praia Vermelha
Rio de Janeiro - RJ CEP 22290-270

Barreira, Luciano de Souza
Ferramenta de streaming de jogos de futebol de robôs da categoria RoboCup Small Size League / Luciano de Souza Barreira, Matheus Bortolini de Oliveira, orientado por Paulo Fernando Ferreira Rosa - Rio de Janeiro: Instituto Militar de Engenharia, 2019.

31p.: il.

Projeto de Fim de Curso (graduação) - Instituto Militar de Engenharia, Rio de Janeiro, 2019.

1. Curso de Graduação em Engenharia de Computação - projeto de fim de curso. 1. Streaming. 2. gRPC. 3. Small Size League. 4. RoboCup. I. Rosa, Paulo Fernando Ferreira . II. Título. III. Instituto Militar de Engenharia.

INSTITUTO MILITAR DE ENGENHARIA

LUCIANO DE SOUZA BARREIRA
MATHEUS BORTOLINI DE OLIVEIRA

FERRAMENTA DE STREAMING DE JOGOS DE FUTEBOL DE ROBÔS DA CATEGORIA ROBOCUP SMALL SIZE LEAGUE

Projeto de Fim de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Engenheiro de Computação.

Orientador: Prof. Paulo Fernando Ferreira Rosa - Ph.D.

Aprovado em 30 de Setembro de 2019 pela seguinte Banca Examinadora:

Prof. Paulo Fernando Ferreira Rosa - Ph.D. do IME - Presidente

Profª. Carla Liberal Pagliari - Ph.D. do IME

Prof. Rafaela Ribeiro Goldschmidt - D.Sc. do IME

Rio de Janeiro
2019

SUMÁRIO

LISTA DE SIGLAS

JSON	JavaScript Object Notation
SSL	Small Size League
LARC	Latin American Robotics Competition
HTTP	Hypertext Transfer Protocol
API	Application Programming Interface
REST	Representational State Transfer

RESUMO

Neste trabalho de conclusão de curso de graduação objetivou-se o desenvolvimento de uma ferramenta de streaming de jogos de futebol de robôs da categoria Small Size League (SSL) da RoboCup. Obteve-se como produto uma plataforma, composta de uma aplicação cliente web e uma aplicação servidor, capaz de transmitir múltiplas partidas da SSL.

ABSTRACT

This undergraduate course work aims the development of a streaming platform for RoboCup's Small Size League (SSL) soccer playing robots matches. As a result, it was developed a platform, composed by both a web client and server applications, able to stream multiple SSL matches.

1 INTRODUÇÃO

São cada vez mais presentes, no cotidiano, tecnologias e aplicações que aproximam pessoas, geram mais formas de entretenimento e tornam mais acessíveis eventos de todos os tipos. Sejam video chamadas em tempo real por *Skype*, assistir a vídeos sob demanda pela *Netflix*, acompanhar uma partida de futebol ao vivo pela internet ou assistir à transmissão de uma corrida de Fórmula 1 do outro lado do mundo.

Nesse cenário, um dos temas mais relevantes é a parte de *streaming*, que segundo Golab e Ozsu (2010), consiste em dados compostos por uma sequência de itens contínua gerada em tempo real. Dado um evento ao vivo, surgem possibilidades de aplicações baseadas em *streaming* para representar esse evento de modo que qualquer pessoa possa acompanhar os acontecimentos pela Internet por meio da reconstrução desses dados.

1.1 MOTIVAÇÃO

No contexto do Instituto Militar de Engenharia, a equipe de futebol de robôs - RoboIME - participa de diversas competições nacionais e internacionais de robótica anualmente. Dentre estas competições, há a categoria Small Size League (SSL) da competição mundial RoboCup de futebol de robôs autônomos, em que não existe uma plataforma de transmissão de partidas que permita, eficientemente, que interessados ao redor do mundo acompanhem amistosos, partidas da RoboCup e da LARC - competição latino-americana de robótica. A demanda latente da comunidade SSL por uma plataforma padrão de *streaming* motivou a criação deste projeto.

O escopo deste projeto busca criar uma ferramenta que torne possível, dentro do âmbito e magnitude das competições, transmissões de partidas de futebol de robôs da SSL, além de permitir que desenvolvedores utilizem-na, também, como ferramenta de desenvolvimento de softwares de inteligência artificial que controlem os robôs.

1.2 OBJETIVO

Este Projeto Final de Curso tem como objetivo a implementação de duas aplicações web para a comunidade RoboCup: (i) servidor de *streaming* - operável com poucos comandos; (ii) cliente - com reprodução das partidas.

Esse objetivo pode ser ilustrado pela figura 1.1 a seguir, onde temos diversas partidas distintas ocorrendo simultaneamente. O servidor capta os dados de todas elas e realiza a transmissão pela internet para que os clientes possam acompanhar a partida do seu interesse.

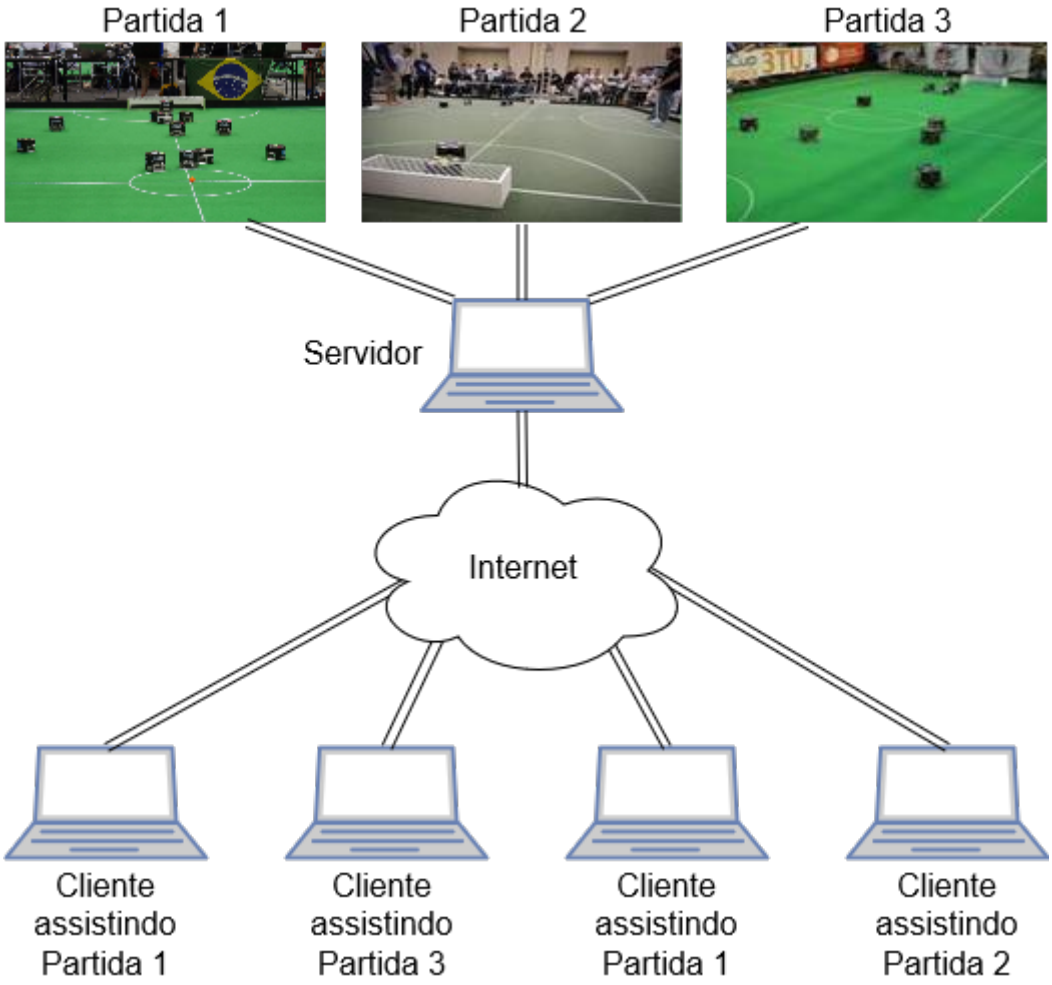

FIG. 1.1: Imagem ilustrativa do objetivo

1.3 METODOLOGIA

Para a realização desse trabalho, foram analisadas técnicas de codificação e decodificação de dados e possíveis tecnologias para o desenvolvimento do servidor, como escolha entre protocolo HTTP/2 ou HTTP/1.1 e protocolos REST ou gRPC de comunicação entre cliente e servidor, visando mitigar os problemas envolvidos numa transmissão em tempo real e, consequentemente, proporcionar uma melhor experiência ao usuário final. Além disso, também foram realizados testes de desempenho de servidor.

1.4 ESTRUTURA

O texto relata todo o planejamento e desenvolvimento dos produtos cliente e servidor de *streaming*. São descritos conceitos necessários para compreensão da arquitetura do sistema bem como as etapas de desenvolvimento do projeto na seguinte estrutura:

- Capítulo 2: Fundamentação teórica e decisões de tecnologias.
- Capítulo 3: Planejamento da arquitetura do projeto.
- Capítulo 4: Desenvolvimento detalhado de todas as partes tanto do servidor quanto do cliente.
- Capítulo 5: Resultados do testes.
- Capítulo 6: Conclusão e possibilidades de melhorias e novas ferramentas para futuros projetos.
- Capítulo 7: Referências bibliográficas utilizadas no projeto.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, introduz-se o contexto de competições da RoboCup SSL e discute-se possíveis tecnologias a serem utilizadas no desenvolvimento de uma plataforma de *streaming* de partidas de futebol de robôs.

Alguns fatores críticos comuns a aplicações *streaming* de grandes eventos ao vivo, tais como o baixo *delay* e alta taxa de transmissão, são restrições relaxadas dentro do escopo este trabalho, uma vez que o produto desenvolvido objetiva a solução de um problema específico demandado por uma comunidade. Isto permite uma maior liberdade para a decisão das tecnologias envolvidas.

2.1 ROBOCUP SMALL SIZE LEAGUE

A RoboCup tem como objetivo realizar, em 2050, uma partida justa de futebol entre robôs e os então campeões mundiais de futebol da FIFA. Dentre as competições propostas como objeto de pesquisa para atingir tal objetivo, a fim de estimular o desenvolvimento em inteligência artificial e em controle, a categoria *Small Size League* (SSL) é composta de competições em que duas equipes de 4 a 11 robôs reais não-humanoides de até 180mm de diâmetro competem autonomamente em uma partida, consoante regras fieis às de futebol convencional (ROBOCUP, 2019).

Durante uma partida de futebol SSL, um software de visão computacional *SSL-Vision*, conectado a 1 até 8 câmeras, apontadas a seções de um campo, extrai das câmeras informações sobre o posicionamento de robôs e bolas em campo e então realiza uma transmissão *multicast* de dados via UDP na rede local da competição. A partir destes dados transmitidos, equipes conectadas à rede local da competição podem utilizar-se das informações sobre a pose - posição e orientação - de robôs de cada time e da posição de possíveis bolas presentes no campo para controlar seus respectivos robôs.

Além do *SSL-Vision*, o software juiz *SSL-Refbox* é responsavel por transmitir, também via UDP multicast, comandos de controle e metadados relativos ao estado da partida. A partir do *SSL-Refbox* cada equipe é capaz de definir o comportamento dos robôs em situações do futebol como de pênalti, lateral e tiro de meta. Dados da partida como placar, nome das equipes e quantidade de cartões amarelos também são transmitidos pelo juiz.

Tanto o funcionamento da visão, quanto do juiz podem ser visualizados pela figura 2.1 a seguir.

A partir dos dados transmitidos na rede, é possível empacotá-los e distribuí-los em tempo real a clientes que desejem assistir a uma partida.

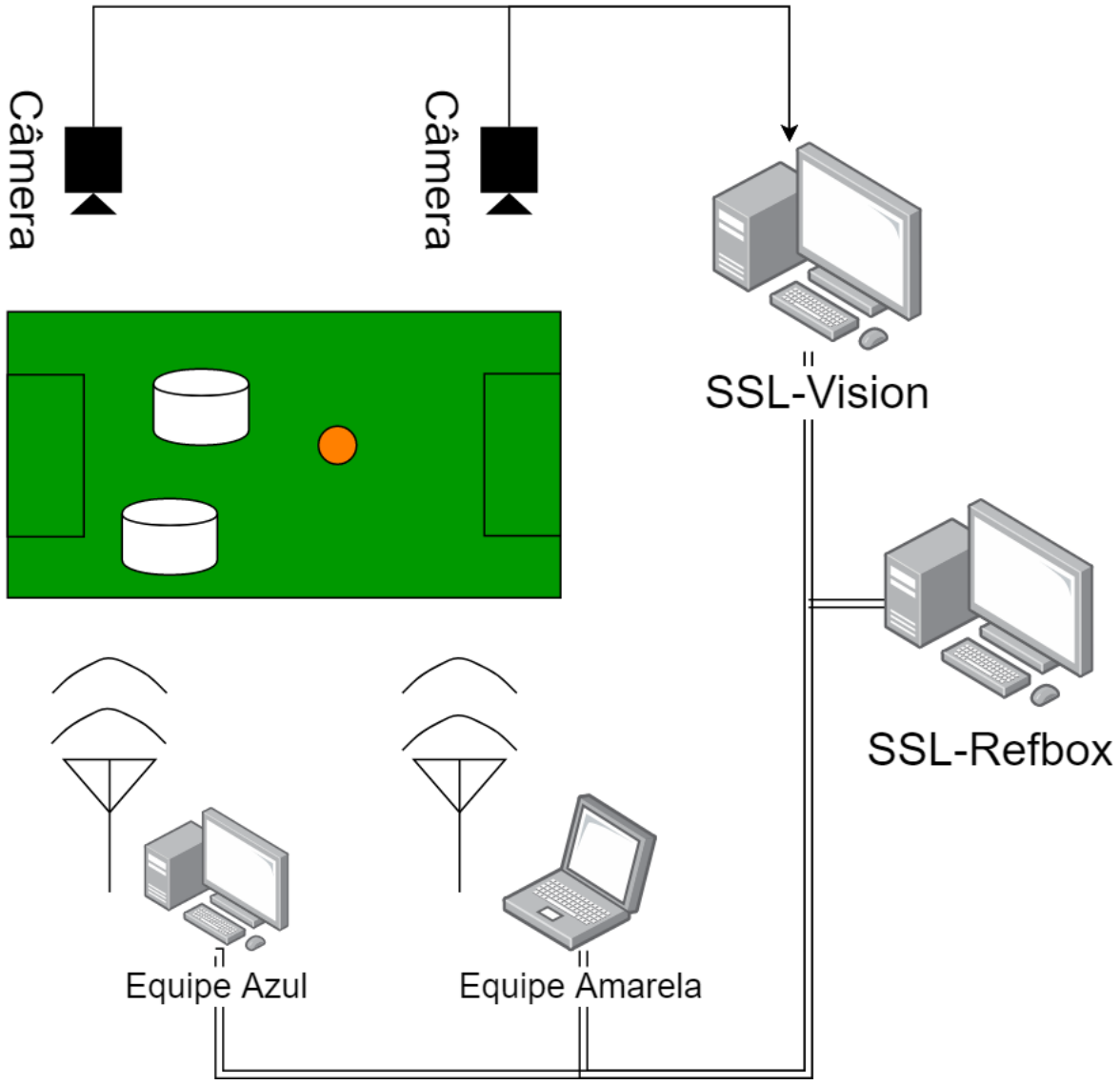

FIG. 2.1: Imagem ilustrativa da transmissão de dados da visão e do juiz

2.2 gRPC

O sistema de chamadas de procedimento remoto gRPC, construído sobre HTTP 2, utiliza como interface de descrição de linguagem o método Protocol Buffers (protobuf), desenvolvido pela empresa Google (GRPC, 2019). Tal protocolo, utilizado também na comunicação dos softwares de visão e do juiz da SSL, constitui de um método de serialização

de dados extensível, agnóstico quanto a linguagens de programação e a plataformas.

A partir de uma descrição única de pacotes e serviços gRPC, é possível gerar bibliotecas a partir das quais desenvolvem-se *stubs* para transmissão de dados entre aplicações cliente e servidor. As transmissões gRPC têm suporte a autenticação, fluxo bidirecional de dados, controle de fluxo, chamadas bloqueantes e não bloqueantes de procedimentos e métodos de cancelamento e *timeout* de transmissão.

2.3 REST

A transferência de estado representacional (REST) surgiu nos anos 2000, criada por um dos principais autores do protocolo HTTP, Roy Fielding, e pode ser encarado como um estilo de arquitetura ou um conjunto de regras que facilita a comunicação entre sistemas e cuja implementação possibilita a independência entre aplicações cliente e servidor.

O padrão REST se baseia em recursos comuns às aplicações e nos verbos utilizados pelo protocolo HTTP para definir como o destinatário daquela requisição deve interpretar e processar o recurso alvo. Por se basear no protocolo HTTP, as requisições no formato REST vêm acompanhadas dos devidos cabeçalhos, que também contém informações relevantes para o servidor, como por exemplo a especificação do recurso e o formato do corpo da requisição, quando necessário, assim como a ação associada (FIELDING, 2000).

2.4 COMPARAÇÃO gRPC E REST

Nesta seção, comparam-se os sistemas gRPC e REST de comunicação entre aplicações cliente e servidor.

2.4.1 COMPARAÇÃO TEÓRICA HTTP 2 VS HTTP 1.1

Uma das diferenças principais entre ambos os protocolos está na montagem dos cabeçalhos. No HTTP 1.1, em cada requisição é enviado um novo cabeçalho completo no formato de texto. Já no HTTP 2, ocorre um reaproveitamento dos cabeçalhos anteriores, só sendo enviados os cabeçalhos que sofreram alterações. Além disso, no HTTP 2 os cabeçalhos são enviados em formato binário e comprimidos, o que no caso médio implica em menor sobrecarga e maior eficiência no uso da rede e eliminando problemas de segurança relacionados ao formato de texto em claro do HTTP 1.1.

Além disso, o protocolo HTTP 2 permite múltiplas requisições assíncronas do cliente para o servidor sobre uma única conexão TCP através da multiplexação de pacotes. Para

ter esse mesmo efeito no HTTP 1.1, é necessário abrir diversas conexões TCP, uma vez que o protocolo é sequencial; portanto, só permite uma requisição por vez. Essa diferença pode ser ilustrada pela figura 2.2.

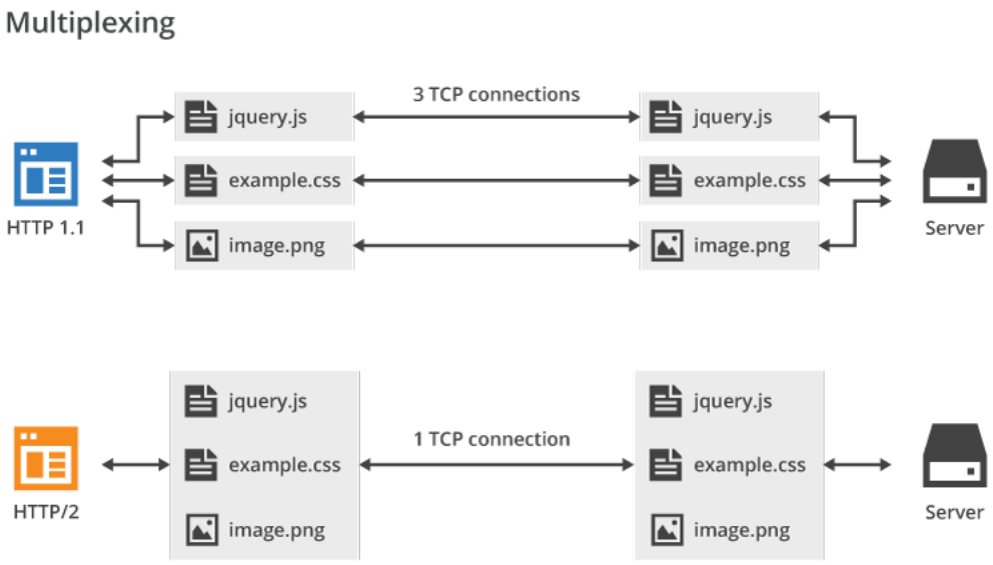

FIG. 2.2: Imagem ilustrativa para múltiplas requisição no HTTP 2 e no HTTP 1.1 (CLOUDFLARE, 2015)

O HTTP 1.1 com *pipelining* (FIELDING; RESCHKE, 2014a) foi uma adaptação para lidar com essa questão. A técnica possibilitava que múltiplas requisições fossem enviadas pelo protocolo HTTP por uma única requisição TCP, porém as respostas do servidor deveriam estar na mesma ordem em que as requisições foram recebidas. Além dessa limitação, o *pipelining* permitia a ocorrência do *HOL Blocking* na camada de aplicação. Essa questão só foi resolvida de fato com a criação do protocolo HTTP 2, que fez com que o HTTP 1.1 com *pipelining* deixa-se de ser suportado pelos navegadores (MDN WEB DOCS, 2017).

Uma outra inovação apresentada pelo HTTP 2 em relação ao HTTP 1.1 é a possibilidade de priorização de recursos. É possível indicar para o navegador quais arquivos são mais críticos à aplicação por meio de priorização numérica, implicando numa maior eficiência e possibilitando uma melhor experiência para aplicações Clientes.

Provavelmente, a nova funcionalidade apresentada pela versão 2 com o maior impacto sobre a nossa aplicação é o *Server Push*. A ideia dessa funcionalidade é o servidor preventivamente poder enviar para a aplicação cliente os recursos que ainda não foram requisitados, mas que serão necessários num futuro próximo, baseado nas requisições feitas

até então pelo cliente. Deste modo, quando o navegador necessitar desse recurso, não será necessário aguardar o processamento de um requisição completa. Este recurso também facilita o projeto e implementação de protocolos de fluxo de dados bi-direcional, como o gRPC. O *Server Push* pode ser utilizado em conjunto com técnicas de otimização de *streaming* (BARBOZA et al., 2015) a fim de melhorar a experiência do usuário final.

O HTTP 2, apesar de ainda pouco explorado pela indústria de software, pode apresentar maior velocidade que HTTP 1.1 (DE SAXCÉ et al., 2015). Esse diferença é ilustrada na figura 2.3, onde temos o número de requisições necessárias no eixo x e o tempo de carregamento da página no eixo y.

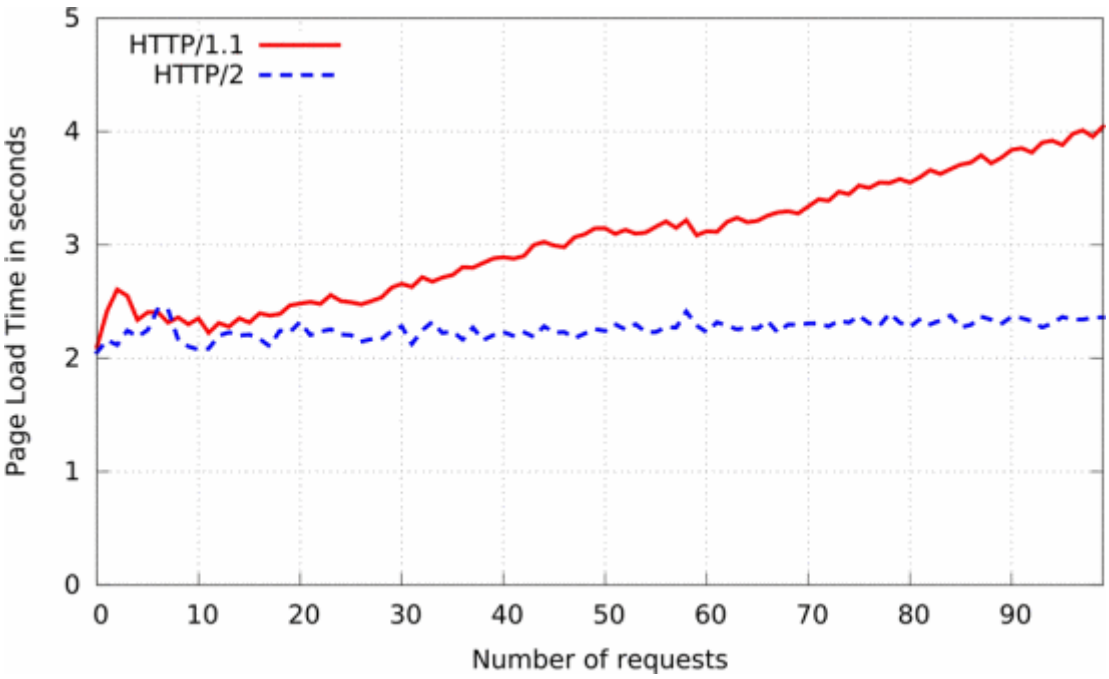

FIG. 2.3: Gráfico comparativo de tempo de carregamento entre HTTP 2 e HTTP 1.1 (DE SAXCÉ et al., 2015)

A tabela 2.1 a seguir resume os principais pontos da comparação entre os protocolos HTTP 2 (BELSHE et al., 2015) e HTTP 1.1 (FIELDING; RESCHKE, 2014b).

TAB. 2.1: Tabela comparativa entre HTTP 2 e HTTP 1

Protocolo	HTTP 2	HTTP 1.1
Envio de Cabeçalhos	Envia somente alterações em relação ao último	Sempre envia cabeçalhos completos
Formato do cabeçalho	Binário e comprimido	Texto em claro
Modo de requisição	Assíncrono	Síncrono
Multiplexação de pacotes	Única conexão TCP	Necessita de várias conexões para o mesmo efeito
Suporte a *Server Push*	Suporta	Nao suporta

2.4.2 COMPARAÇÃO ENTRE gRPC E REST

Para a Interface de Programação de Aplicação do servidor, poderiam ser utilizados gRPC ou REST, sendo que, dadas as vantagens de se utilizar o protocolo HTTP 2, não existem razões por padrão para gRPC ser mais eficiente do que REST. A vantagem do gRPC se deve às ferramentas e integrações que possibilitam uma maior eficiência quando comparado ao REST.

Uma das diferenças se dá no formato do corpo da requisição. Enquanto REST tipicamente utiliza JSON dado o ecossistema e aplicações web desenvolvidos ao longo do tempo com este padrão, mesmo sem ser um formato obrigatório, gRPC é otimizado para a utilização de Protobuf, método de serialização que apresenta maior eficiência no uso de dados estruturados (MAEDA, 2012). Isto pode ser ilustrado pela figura 2.4, onde temos no eixo x o número de elementos de uma lista que faz parte do objeto a ser serializado e o tempo de execução deste processo no eixo y.

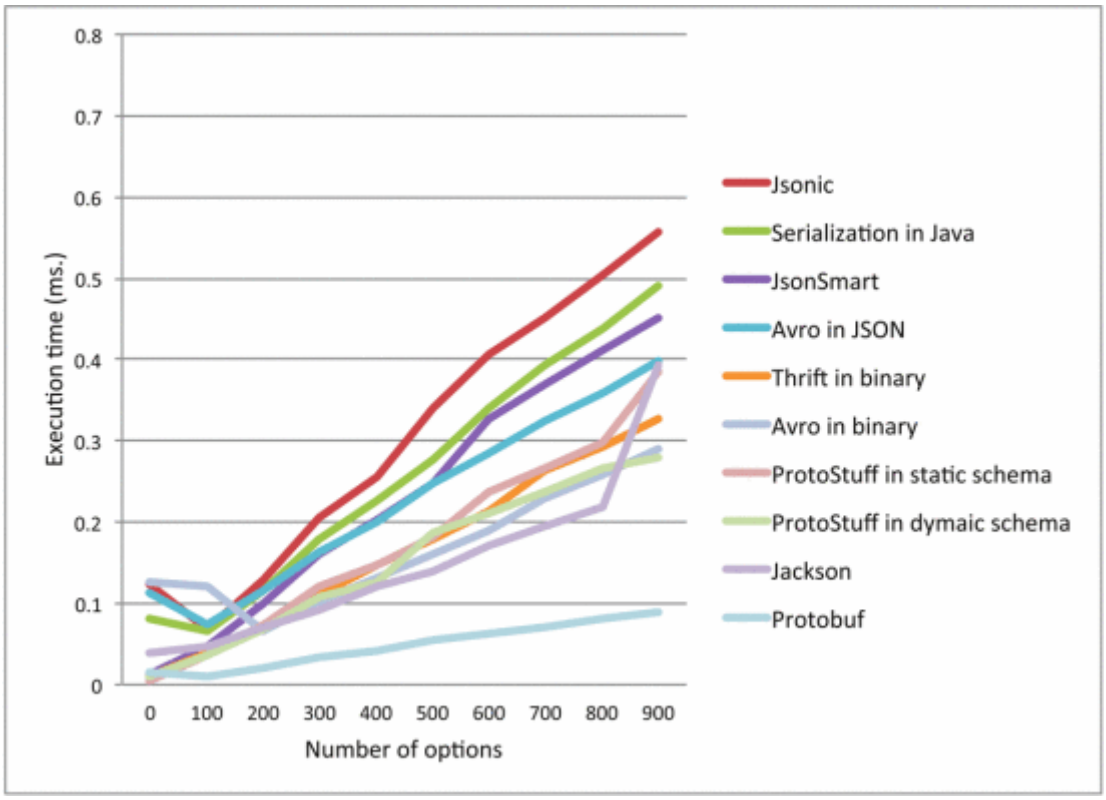

FIG. 2.4: Gráfico comparativo de tempo de execução da serialização de diversas bibliotecas (MAEDA, 2012)

Uma das desvantagens do gRPC frente ao REST se dá em relação ao suporte por parte dos navegadores. Como gRPC ainda não é bem suportado por navegadores, isso faz com que ele não seja a melhor solução para aplicações servidor que sejam feitas para proverem serviços externos.

A tabela 2.2 a seguir resume os principais pontos da comparação entre os protocolos gRPC (GRPC, 2019) e REST (FIELDING, 2000).

TAB. 2.2: Tabela comparativa entre gRPC e REST

API	gRPC	REST
Suporte a protocolos HTTP	HTTP 2, apenas	HTTP 1.1 e HTTP 2
Formato do corpo da requisição	Binário (Protobuf)	JSON
Suporte para *streaming* bidirecional	Suporta	Não suporta
Serialização de dados inclusa	Cliente e servidor	Não inclui
Balanceamento de carga de Servidor	Suporta	Não suporta
Amplitude de suporte de navegadores	Suporte restrito	Amplo suporte

2.4.3 TESTE DE DESEMPENHO gRPC E REST

Um teste comparativo foi desenvolvido na linguagem de programação Go (BARREIRA, 2019) para medir o desempenho das abordagens gRPC e REST quanto ao uso de CPU e de memória RAM. Ambos os testes foram realizados na mesma máquina Linux sob um processador Intel(R) Core(TM) i5-8350U CPU @ 1.70GHz, de 8 núcleos.

Neste teste, são criadas aplicações cliente e servidor em rede local e efetuadas 20000 requisições a um servidor juiz da SSL. Dentre as métricas disponíveis pela biblioteca padrão de teste da linguagem Go, obtém-se o tempo de execução médio, número de bytes de memória RAM utilizados e quantidade de alocações por operação(op) e os resultados podem ser observados na tabela 2.3 a seguir.

TAB. 2.3: Testes de desempenho de gRPC e REST

Teste	tempo(ns)/op	bytes/op	alocações/op
gRPC	67888	10194	183
REST	75925	9133	111

Apesar de os resultados próximos em tempo, memória e quantidade de alocações, o uso de gRPC não representa perdas significativas de desempenho quanto à abordagem REST.

3 PLANEJAMENTO DA ARQUITETURA DO PROJETO

O planejamento da arquitetura do projeto pode ser dividido em duas etapas para facilitar a explicação: (i) primeira etapa considerando uma única partida ocorrendo; (ii) segunda etapa considerando múltiplas partidas simultâneas.

Para ambas as etapas, o servidor estará fisicamente localizado no local onde a competição estiver sendo disputada e estará conectado na rede onde estarão sendo transmitidos dados do software de visão computacional e do juiz.

3.1 PARTIDA ÚNICA

Neste caso, a responsabilidade do servidor será de processar, empacotar e distribuir os dados da partida através do protocolo gRPC para as aplicações clientes conectadas. A aplicação cliente por sua vez será responsável por reproduzir as situações do jogo baseado nas informações transmitidas pelo servidor.

A figura 3.1 serve como ilustração lógica desta arquitetura de uma única partida.

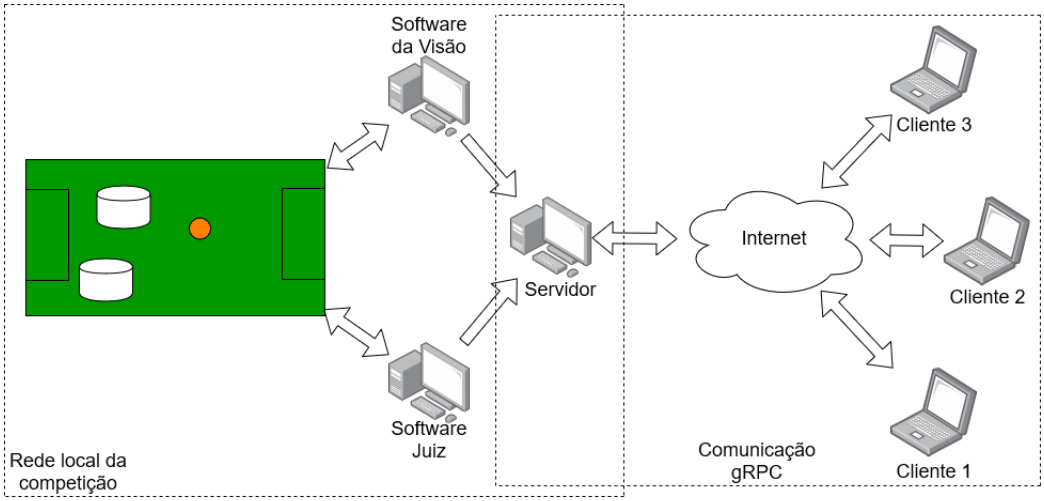

FIG. 3.1: Imagem ilustrativa da arquitetura de uma única partida

3.2 MÚLTIPLAS PARTIDAS

Para múltiplas partidas simultâneas são necessários alguns recursos adicionais. Do lado do servidor passa a ser necessário agrupar os dados da rede local da competição de acordo com a partida a qual ele se refere antes de realizar o processamento, empacotamento e distribuição desses dados. Além disso, também passa a ser necessário que o servidor informe ao cliente uma lista de todas as partidas disponíveis para serem acompanhadas.

Já do lado do cliente, passa a ser necessário que seja informado a partida de interesse para que somente os dados correspondentes sejam transmitidos. Desse modo, a arquitetura para múltiplas partidas pode ser representada pela figura 3.2.

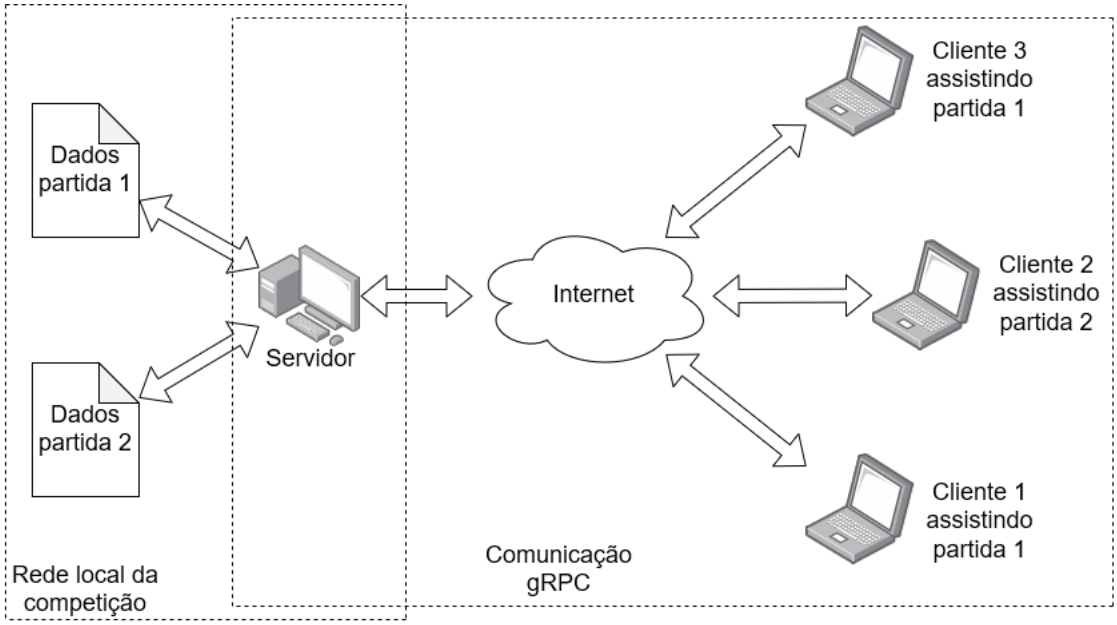

FIG. 3.2: Imagem ilustrativa da arquitetura para múltiplas partidas

4 DESENVOLVIMENTO DO PROJETO

O desenvolvimento do projeto pode ser dividido em duas partes, tal como o objetivo deste trabalho: (i) parte relativa a aplicação servidor; (ii) parte relativa a aplicação cliente.

4.1 SERVIDOR

A implementação do servidor foi feita utilizando a linguagem de programação Go devido às ferramentas providas pela linguagem para implementação de modelos de concorrência, à familiaridade e à produtividade com o seu uso. O servidor consta de três módulos principais: Visão, Juiz e Servidor gRPC. Cada módulo é executado concorrentemente em Goroutines - modelo de rotinas concorrentes de Go - distintas.

4.1.1 VISÃO

O módulo de Visão estabelece uma comunicação UDP com múltiplas fontes de pacotes de Quadros de Detecção, seja um processo do software de visão computacional *SSL-Vision* ou de um simulador SSL, como o *grSim*.

Pacotes de Quadros de Detecção são armazenados em um *buffer* de tamanho fixo, atualizado à taxa em que os pacotes são recebidos, descartando o pacote mais antigo do *buffer* para a entrada do pacote mais recente. Cada Quadro de Detecção contém informações de bolas - como posição - e de robôs - como posição, orientação, time e ID - detectados por uma câmera. Este módulo mantém um conjunto de *buffers* em que cada *buffer* contém quadros de detecção relativos a uma câmera.

4.1.2 JUIZ

O módulo de Juiz estabelece uma comunicação UDP com o software *SSL-Refbox*, responsável por transmitir informações de uma partida como os comandos do Juiz, estado da partida, eventos do jogo e dados das equipes competidoras.

O último pacote recebido do Juiz é atualizado à taxa de recepção e transmitido ao cliente quando requisitado. Há um pacote por partida corrente.

4.1.3 SERVIDOR gRPC

O módulo de Servidor gRPC configura um servidor gRPC capaz de satisfazer as seguintes requisições:

- Sequência de Quadros de Detecção de uma determinada partida.
- Pacote de dados de Juiz de uma determinada partida.
- Informações de conjuntos de partidas ativas.

Este módulo mantém comunicação concorrente com os módulos de Juiz e de Visão, acessando as informações necessárias para atender requisições.

4.2 CLIENTE

A implementação do cliente foi realizada utilizando a linguagem de programação NodeJS e o *framework* React por conta da grande popularidade e suporte deste *framework* e sua mecânica de funcionamento - que permite dividir uma *View* nos chamados componentes do React, facilitando o desenvolvimento da aplicação em questão.

A aplicação em si consiste em somente uma página que apresenta algumas informações ao redor do campo, onde a reprodução da partida é renderizada. Para a construção do campo foi utilizada a tag html SVG como base, cujas partes do campo são componentes filhos desse elemento. Para auxiliar no detalhamento desta estrutura e de seus diversos componentes, tomaremos como referência a figura 4.1 que exemplifica uma situação inicial de jogo.

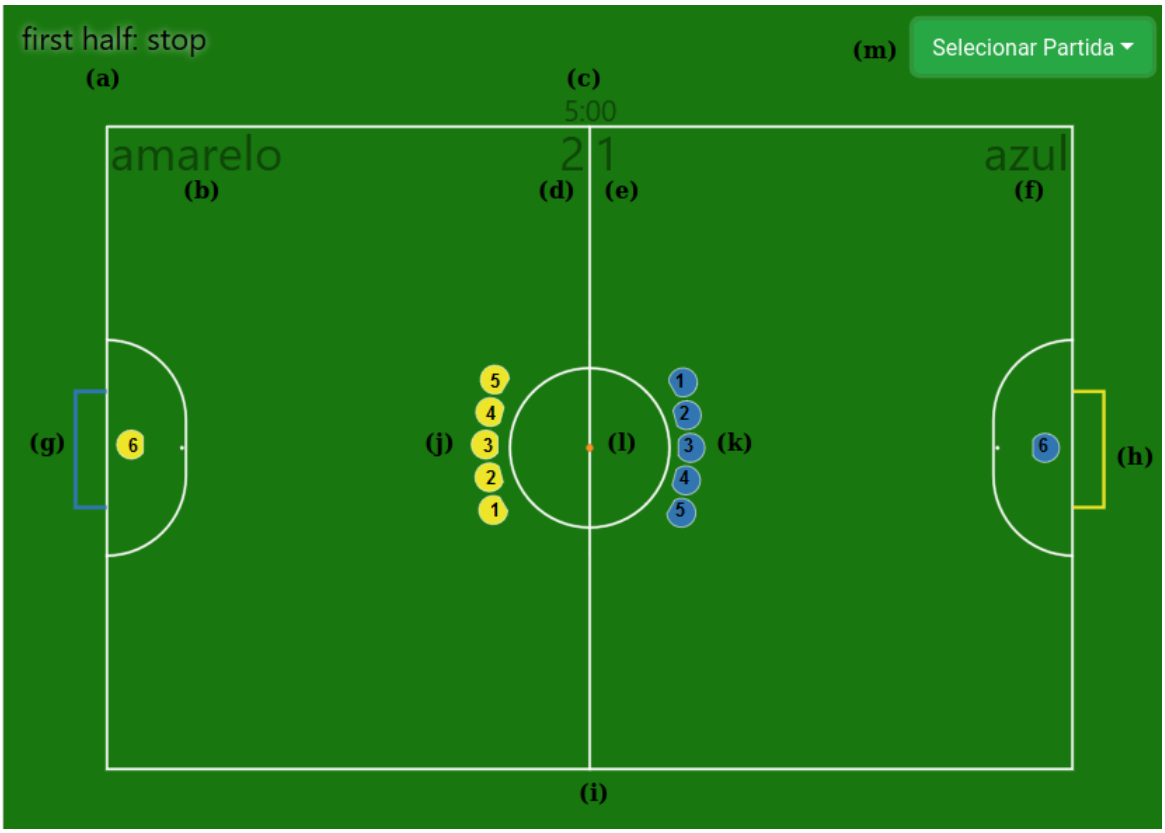

FIG. 4.1: Imagem ilustrativa de uma formação inicial

Descrição de cada componente:

- (a): Componente informativo que recebe como parâmetros códigos do estágio do jogo e do comando do juiz e faz o mapeamento para transformar no texto correspondente conforme tabela 8.1 do **Apêndice I** e tabela 9.1 do **Apêndice II** respectivamente.
- (b) e (f): Componente que recebe como parâmetro o nome do time correspondente a respectiva cor.
- (c): Componente que recebe como parâmetro o restante da partida.
- (d) e (e): Componente que recebe como parâmetro o número de gols marcado pelo respectivo time.
- (g): Componente representando o gol a ser atacado pelo time azul.
- (h): Componente representando o gol a ser atacado pelo time amarelo.

- (i): Componente que recebe como parâmetros todas as dimensões do campo para representar todas as linhas brancas - retângulo exterior, linha central, áreas e círculo central.
- (j) e (k): Componente robô representando cada um dos seis robôs de cada time que recebe como parâmetros cor, texto, ângulo de orientação e coordenadas x e y.
- (l): Componente representando a bola que recebe como parâmetros as coordenadas x e y.
- (m): Componente *dropdown* com as opções de partidas para serem selecionadas.

5 RESULTADOS DOS TESTES

Foram realizados testes de carga nesta primeira versão do servidor descrita pelo capítulo 4.1 e em versões posteriores com melhorias. Em ambos os testes o processador utilizado na máquina de teste foi um Intel(R) Core(TM) i5-8350U CPU @ 1.70GHz, de 8 núcleos, com uma memória RAM DDR4 de 8192 MB.

5.1 TESTE DE CARGA NO SERVIDOR DE PARTIDA ÚNICA

A tabela 5.1 a seguir mostra o consumo médio percentual de CPU e de memória RAM por quantidade de clientes no processo teste para 10, 50 e 100 clientes concorrentes. O processo teste cria uma quantidade de clientes que efetuam chamadas gRPC ao servidor a uma determinada taxa. Neste teste, cada cliente efetua uma requisição por 100 milissegundos. Os dados de consumo de CPU e de memória RAM foram amostrados a cada segundo durante 60 segundos. O tamanho de *buffer* da Quadros de Detecção utilizado foi de um quadro apenas.

Neste teste, o servidor realiza *streaming* de uma partida apenas.

TAB. 5.1: Teste de carga no servidor

Clientes	% CPU	% Memória RAM
10	9.95	0.1
50	36.2	0.2
100	72.2	0.3

Os resultados do teste mostram que há espaço para otimização de consumo de CPU e será base comparativa para testes futuros. Algumas sugestões de melhorias que podem diminuir o consumo de CPU são:

- Aplicação de algoritmos de filtragem de pacotes vindos do *SSL-Vision*, com objetivo de reduzir a redundância entre pacotes consecutivos, evitando a transmissão de quadros com alta taxa de correlação entre dados.
- Fusão de Quadros de Detecção de câmeras distintas em um mesmo intervalo de tempo, uma vez que cada Quadro de Detecção representa informações vindas de uma câmera apenas.

- Uso de buffer de Quadros de Detecção na aplicação cliente, diminuindo a necessidade de manter uma taxa alta de requisições.

5.2 TESTE DE CARGA NO SERVIDOR MULTI-PARTIDAS COM FUSÃO DE QUADROS

Após implementação de algoritmo de Fusão de Quadros de Detecção conforme proposto na seção 5.1, foram re-executados testes de carga no servidor descritos na seção 5.1.

5.2.1 ALGORITMO DE FUSÃO DE QUADROS

Para a fusão de quadros, são armazenados os últimos quadros de detecção de cada câmera e fundidos em um quadro único para envio ao cliente. Para conflitos de robôs de mesmo identificador detectados por câmeras distintas, o robô enviado no quadro é calculado pela a média das poses dos robôs conflitantes.

5.2.2 RESULTADOS

Neste teste, o servidor realiza *streaming* de duas partidas simultaneamente, em que cada metade dos clientes faz requisições a partidas distintas. A tabela 5.2 a seguir mostra o resultado deste teste. A metodologia de teste utilizada é a mesma descrita na seção 5.1.

TAB. 5.2: Teste de carga no servidor com fusão de quadros de detecção

Clientes	% CPU	% Memória RAM
10	5.3	0.2
50	26.4	0.2
100	45.1	0.2

Os resultados do teste mostram melhoria significativa no desempenho do servidor após implementação do algoritmo de Fusão de Quadros de Detecção, ainda que restrito ao último quadro de cada câmera, chegando a uma melhoria de 60% em desempenho de CPU para 100 clientes concorrentes.

6 CONCLUSÃO

Este trabalho teve como objetivo o desenvolvimento de duas aplicações, cliente e servidor, para a comunidade RoboCup que possibilitem o acompanhamento das partidas de futebol de robô realizadas durante as competições. Após um estudo inicial de tecnologias disponíveis para o desenvolvimento de um servidor de *streaming*, optou-se pelo protocolo *gRPC* e foi desenvolvido um produto que permite reproduzir as situações de um jogo baseado nos pacotes transmitidos durante a partida.

O produto final provê suporte a múltiplas partidas simultâneas em que cada cliente tem a opção de escolher qual prefere acompanhar. Além disso, também foi implementada a fusão de quadros no servidor a fim de diminuir o consumo de CPU, chegando a uma melhoria de até 60% em relação à versão sem esta técnica.

Por fim, espera-se que o produto gerado seja bem aceito e adotado pela comunidade RoboCup. Como propostas para trabalhos futuros, sugere-se a implementação das seguintes melhorias já levantadas neste trabalho: algoritmos de filtragem de pacotes no servidor e uso de *buffer* para quadros de detecção no cliente -, que também melhorariam o desempenho de CPU do servidor. Uma vez que este projeto está disponível como código aberto sob licença MIT, espera-se também que possíveis demandas da comunidade, como adição de vídeo com *overlay* de informações da partida, sejam incorporadas ao projeto.

7 REFERÊNCIAS BIBLIOGRÁFICAS

BARBOZA, D. C.; MUCHALUAT-SAADE, D. C. ; CLUA, E. W. G. A real-time game streaming optimization technique based on layer caching. In: 2015 12TH ANNUAL IEEE CONSUMER COMMUNICATIONS AND NETWORKING CONFERENCE (CCNC), 1., 2015. **Anais**... [S.l.: s.n.], 2015, p. 714–719.

L. BARREIRA. gRPC REST Bechmark. Disponível em: <https://github.com/lucbarr/grpc-rest-benchmark>. Acesso em: 22 jul. de 2019.

M. BELSHE AND R. PEON AND M. THOMSON. Hypertext Transfer Protocol Version 2 (HTTP/2). Disponível em: <https://tools.ietf.org/html/rfc7540>. Acesso em: 22 jul. de 2019.

CLOUDFLARE. HTTP/2 For Web Developers. Disponível em: <https://blog.cloudflare.com/http-2-for-web-developers/>. Acesso em: 22 jul. de 2019.

DE SAXCÉ, H.; OPRESCU, I. ; CHEN, Y. Is http/2 really faster than http/1.1?. In: 2015 IEEE CONFERENCE ON COMPUTER COMMUNICATIONS WORKSHOPS (INFOCOM WKSHPS), 1., 2015. **Anais**... [S.l.: s.n.], 2015, p. 293–299.

R. FIELDING AND J. RESCHKE. Hypertext Transfer Protocol (HTTP/1.1). Disponível em: <https://tools.ietf.org/html/rfc7230>. Acesso em: 14 oct. de 2019.

R. FIELDING AND J. RESCHKE. Hypertext Transfer Protocol (HTTP/1.1). Disponível em: <https://tools.ietf.org/html/rfc7231>. Acesso em: 22 jul. de 2019.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. Tese (Doutorado em Arquitetura de Software) – University of California, Irvine, 2000.

GOLAB, L.; OZSU, M. T. **Data Stream Management**. [S.l.]: Morgan Claypool, 2010. ISBN 9781608452736.

GRPC. gRPC. Disponível em: <https://www.grpc.io/>. Acesso em: 22 jul. de 2019.

MAEDA, K. Performance evaluation of object serialization libraries in xml, json and binary formats. In: 2012 SECOND INTERNATIONAL CONFERENCE ON DIGITAL INFORMATION AND COMMUNICATION TECHNOLOGY AND IT'S APPLICATIONS (DICTAP), 1., 2012. **Anais**... [S.l.: s.n.], 2012, p. 177–182.

MDN WEB DOCS. Connection management in HTTP/1.x. Disponível em: <https://wiki.developer.mozilla.org/en-US/docs/Web/HTTP/Connection_management_in_H Acesso em: 14 oct. de 2019.

ROBOCUP. RoboCup Federation Official Website. Disponível em: <https://www.robocup.org/>. Acesso em: 22 jul. de 2019.

8 APÊNDICE I: MAPEAMENTO ENTRE CÓDIGO E ESTÁGIO DO JOGO

TAB. 8.1: Tabela descrevendo os possíveis estágios de uma partida

Código	Estágio	Descrição
0	pre game	Primeiro tempo prestes a começar
1	first half	Primeiro tempo em andamento
2	half time	Intervalo entre primeiro e segundo tempos
3	pre second half	Segundo tempo prestes a começar
4	second half	Segundo tempo em andamento
5	extra time break	Intervalo entre o tempo normal e a prorrogação
6	pre extra first half	Primeiro tempo da prorrogação prestes a começar
7	extra first half	Primeiro tempo da prorrogação em andamento
8	extra half time	Intervalo entre primeiro e segundo tempos da prorrogação
9	pre extra second half	Segundo tempo da prorrogação prestes a começar
10	extra second half	Segundo tempo da prorrogação em andamento
11	penalty shootout break	Intervalo entre a prorrogação e os pênaltis
12	penalty shootout	Pênaltis em andamento
13	post game	Fim de jogo

9 APÊNDICE II: MAPEAMENTO ENTRE CÓDIGO E COMANDO DO JUIZ

TAB. 9.1: Tabela descrevendo os possíveis comando do juiz

Código	Comando	Descrição
0	halt	Todos os robôs devem ficar parados
1	stop	Robôs devem manter distância de 50 cm da bola
2	start	Uma falta ou um pênalti será cobrado
3	force start	Bola em jogo
4	prepare kickoff yellow	Time amarelo deve se posicionar para cobrança de uma falta
5	prepare kickoff blue	Time azul deve se posicionar para cobrança de uma falta
6	prepare penalty yellow	Time amarelo deve se posicionar para cobrança de um pênalti
7	prepare penalty blue	Time azul deve se posicionar para cobrança de um pênalti
8	direct free yellow	Time amarelo deve cobrar uma falta direta
9	direct free blue	Time azul deve cobrar uma falta direta
10	indirect free yellow	Time amarelo deve cobrar uma falta indireta
11	indirect free blue	Time azul deve cobrar uma falta indireta
12	timeout yellow	Pausa técnica pedida pelo time amarelo
13	timeout blue	Pausa técnica pedida pelo time azul
14	goal yellow	Time amarelo acabou de marcar um gol
15	goal blue	Time azul acabou de marcar um gol