PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Uma Arquitetura para um Sistema Distribuído de Controle de Tráfego Inteligente em Tempo Real

Victor Meira Pinto

PROJETO FINAL DE GRADUAÇÃO

CENTRO TÉCNICO CIENTÍFICO - CTC
DEPARTAMENTO DE INFORMÁTICA
Curso de Graduação em Engenharia da Computação

Rio de Janeiro, Dezembro de 2019

VICTOR MEIRA PINTO

Uma Arquitetura para um Sistema Distribuído de Controle de Tráfego Inteligente em Tempo Real

Relatório de Projeto Final, apresentado ao programa **Engenharia da Computação** da PUC-Rio como requisito parcial para a obtenção do título de Engenheiro de Computação

Profa. Noemi Rodriguez
Orientador
Departamento de Informática - PUC-Rio

Rio de Janeiro,
Dezembro de 2019

Agradecimentos

Primeiramente, a meus pais, avó e irmão, pelo apoio moral e financeiro, sem vocês nunca chegaria aqui.

Aos meus amigos, Bernardo, Alexandre e Rafael, pelos dias e noites de companhia durante essa longa jornada de 5 anos e meio, sem vocês, não sei se teria conseguido.

À minha orientadora e coordenadora do curso, Noemi, pela paciência, conselhos e aprendizado que recebi desde que entrei na faculdade

À PUC-Rio por me proporcionar a oportunidade de crescer tanto profissionalmente quanto pessoalmente, jamais tudo que passei para chegar aonde cheguei.

E ao meu gato, Bob, pela fofura e a companhia nos dias mais difíceis.

Resumo

Meira Pinto, Victor. Rodriguez, Noemi. Uma Arquitetura para um Sistema Distribuído de Controle de Tráfego Inteligente em Tempo Real Rio de Janeiro, 2019. Relatório Final de Projeto de Conclusão de Curso – Centro Técnico Científico – CTC, Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

Esse trabalho tem como objetivo estudar, analisar e propor uma arquitetura para um sistema de tráfego inteligente distribuído que pode ser implementado em uma cidade de alto índice de tráfego como o Rio de Janeiro. Dentro da arquitetura, é usada a tecnologia do Redis e assim como foi utilizada a linguagem Lua junto ao framework Löve2D para simulações.

Abstract

Meira Pinto, Victor. Rodriguez, Noemi. An architecture for a Distributed System for Real-Time Intelligent Traffic Control. Rio de Janeiro, 2019. Relatório Final de Projeto de Conclusão de Curso – Centro Técnico Científico – CTC, Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

This project aims to study, analyze and propose an architecture for an distributed intelligent real-time traffic system that could be implemented in a city similar to Rio de Janeiro, with high car traffic. Inside the arquitecture, Redis is present as well as Lua using Löve 2D frameworks for simulations.

Tabela de Conteúdos

I. Introdução

Com o advento das tecnologias de comunicação sem fio entre aparelhos eletrônicos e do barateamento de componentes de processamento e sensores dos mais variados, nos vemos hoje com uma qualidade de vida muito maior do que havia há poucos anos atrás. Trocas de mensagens ao redor do mundo em tempo real, fotos em alta qualidade com câmeras que cabem no nosso bolso, alertas de segurança e clima recebidos por todos em instantes, até mesmo carteira de habilitação em forma digital. Apesar de todos esses avanços, no cenário de mobilidade urbana e de tráfego nas cidades, parece que nada mudou. Sinais dessincronizados, cruzamentos que sempre causam retenções, períodos de espera longos para que haja passagem de pedestres, sem haver algum. Um estudo de 2017 estima que cerca de 35 bilhões de reais são perdidos anualmente com tempo gasto no trânsito da cidade do Rio de Janeiro [1]. No dia a dia, são problemas como esses que diminuem a qualidade de vida dos moradores da cidade, causando estresse e perda de tempo que acabam afetando a saúde e bem-estar de todos.

Alguns trabalhos na área têm procurado usar a tecnologia que está a nossa disposição nas outras áreas das nossas vidas e aplicá-las no trânsito, criando um sistema de controle de tráfego inteligente que diminua o tempo de espera de tanto os pedestres quanto dos motoristas. Um sistema de controle desse, porém, requer uma coesão de diferentes partes e mecanismos de comunicação que sejam capazes de atuar confiavelmente e rapidamente tornando-o difícil de implementar de forma eficaz sem que haja um estudo da estrutura do sistema como um todo. Esse trabalho tem como objetivo entender as dificuldades e envolvimento quanto à infraestrutura de comunicação e armazenamento de histórico e sugerir um modelo de arquitetura distribuída para um sistema de modelo de tráfego inteligente em tempo real de forma distribuída. O trabalho não se propõe a se aprofundar na área de estudo do fluxo do tráfego e áreas relacionadas — estudando fórmulas de melhor funcionamento do trânsito utilizando algoritmos específicos — tanto quanto inovar na área de sistemas de trânsito inteligentes, e sim a utilizar conceitos simples de ambos para o plano do sistema.

Esse projeto foi todo desenvolvido usando Lua, ferramentas em frameworks da linguagem, como o Löve2D, que auxilia na parte gráfica do projeto, e pequenas bibliotecas para Lua como Json[1], Socket[2], etc. que auxiliaram na simplificação do código. A parte da arquitetura utiliza um banco em memória chamado Redis para grande parte da comunicação entre os

[1] https://github.com/rxi/json.lua

[2] http://w3.impa.br/~diego/software/luasocket/

componentes. Todos essas tecnologias podem ser utilizadas em qualquer plataforma. No nosso caso específico, foram utilizadas as plataformas do macOS[3] e também do Ubuntu para alguns testes.

Como esse sistema envolve o desenvolvimento e análise de uma arquitetura para um sistema de tráfego, foram utilizados diversos conceitos ensinados durante o curso, como o de sistemas distribuídos, bancos de dados, redes, algoritmos variados além da própria lógica de programação que também é necessária para se completar um trabalho como esse.

II. Situação Atual

Utilizando o conceito de *Smart City*[4], ideia de uma cidade inteligente onde os diferentes elementos da cidade se comunicam com o objetivo de manter a harmonia e aumentar a coesão entre os cidadãos e o ambiente urbano, seja por via de tratamento da qualidade do ar, controle de posição e rotas de ônibus, monitoramento de luz pela cidade, entre outros, esse projeto propõe uma solução para o alívio do trânsito em metrópoles de alto número de carros. Atualmente, os sistemas de tráfego inteligente não têm sido bem-sucedidos por uma série de fatores, seja por falta de manutenção, ou má implementação. Existem exemplos em Niterói, de sinais inteligentes implantados no começo de 2016 [2] que não ajudaram combater o problema de trânsito na cidade, até piorando-o, de acordo com um jornal [3]. Enquanto isso, o Rio de Janeiro continua em um sistema semafórico pré-programado, implementado em 1995, onde se utiliza planos de tempo fixo, ou seja, todo dia, nos mesmos horários os sinais operam em verde, amarelo e vermelho. Técnicos da CET-Rio são capazes de alterar manualmente tais operações, mas levando a afetar negativamente o sincronismo com os outros sinais. Na Rua Conde do Bonfim, na Tijuca, foi implantado, no começo da década, um modelo de sistema adaptativo, que regulava o tempo de acordo com o fluxo por uso de câmeras, levando a um decréscimo de 10% no tempo de passagem pela rua. [4] Recentemente, no mês de novembro desse ano, 2019, foram adicionados ao centro da cidade de Duque de Caxias um sistema de monitoramento de tráfego que utiliza câmeras para fazer a leitura do fluxo dos carros e conseguir automaticamente ajustar o tempo de abertura do sinal. [5]

---

[3] Sistema Operacional desenvolvido pela Apple para sua linha de computadores pessoais.

[4] Smart cities são projetos nos quais um determinado espaço urbano é palco de experiências de uso intensivo de tecnologias de comunicação e informações sensíveis ao contexto (Internet of Things).

Já no espaço acadêmico, exemplos de sistemas como esses utilizam técnicas de aprendizado de máquina [6][7], ou utilizam um modelo *VANET*[5] [8][9] onde os veículos se comunicam entre si e com a estrada para que o tráfico seja controlado, todos que dependem de um bom investimento.

Também existem propostas de sistemas que se auto organizam utilizando um modelo de tráfego de intervalo microscópico, onde a decisão é tomada por cada interseção independentemente, e, cada previsão é feita dado um intervalo de tempo e um veículo específico, e o algoritmo analisa qual dessas previsões será a mais beneficial para todos. Esses tipos de propostas se baseiam muito em teorias de controle de fluxo de tráfego e fogem do escopo pretendido pelo trabalho. [10]

Apesar dos projetos nessa área não terem decolado, existe uma grande quantidade de sensores que podem ser usados hoje em dia para esse tipo de sistema: sensores infravermelhos ativos e passivos -- utilizados para medições de velocidade, classificação e tamanho do veículo, entre outros; sensores ultrassônicos -- mais utilizado para controle de velocidade, por ser mais barato; sensores vetoriais acústicos passivos -- utilizado também para medição de velocidade e passagens de veículos; loops de indução -- colocados debaixo do asfalto para detectar a presença de carros parado a frente ao sinal; entre muitos outros. [11]

Foi feito um estudo publicado na MDPI (Multidisciplinary Digital Publishing Institute) em 2018 [12], que analisa inúmeros sensores de monitoramento de tráfego em vias para identificar aquelas tecnologias com alto índice de acurácia e com baixo custo em geral. Dentre os mais de 20 sensores e trabalhos acadêmicos analisados, conclui-se que o mais importante nesses tipos de sistema é a utilização de mais de um sensor, aumentando significativamente o índice de acerto e de que existem sim sensores de baixo custo com alta acurácia, apesar de muito desses ainda serem modelos propostos em trabalhos acadêmicos e não disponíveis para compra.

Outra possibilidade que não parece ser explorada nessas implementações é o tratamento de dados na nuvem, que facilita o custo de implementação e manutenção do projeto, além de providenciar um maior poder computacional para se adicionar novos sensores.

[5] Vehicular Ad hoc Network - http://www.rroij.com/open-access/vehicular-ad-hoc-network-vanets-a-review.php?aid=55478

Atualmente, com o advento de inúmeras formas de comunicação entre aparelhos diferentes como o *HTTP*[6], muito utilizado na Web, o *MQTT*[7], na especificação do projeto, fica difícil escolher qual protocolo utilizar. Contudo essa variedade facilita a moldar a comunicação para cada um dos componentes de um modelo de sistema, possibilitando que aparelhos com baixo poder computacional possam se comunicar facilmente com outras partes, que por sua vez podem se comunicar via outro gateway, que priorize a velocidade ou segurança.

III. Proposta e Objetivos

Nesse trabalho investigaremos uma arquitetura adequada para um sistema de controle de tráfego, através do controle de sinais localizados em cruzamentos, utilizando sensores e contexto histórico do fluxo, com intuito de diminuir a congestão de veículos. O sistema será projetado de forma distribuída para que as interseções possam trocar informações e processar dados dos sensores em tempo real de tal forma a reagir às mudanças do trânsito, dando ênfase, principalmente, para o armazenamento das informações recebidas e a troca dessas informações entre cada um dos componentes. O objetivo do trabalho será estudar e desenvolver um modelo de arquitetura que leve em consideração que o sistema deverá reagir em tempo real às situações que estiverem acontecendo, devendo ser distribuído de tal forma a ser altamente escalável, e ser inteligente de conseguir, através de contexto histórico e do estado presente do tráfico, controlar um semáforo com intuito de minimizar a congestão de veículos. As metas do sistema são a diminuição do tempo gasto em cada sinal, diminuição no tempo de espera de pedestres tentando atravessar a via, e redução do nível de acidentes nos cruzamentos.

O sistema proposto seria encarregado de medir e controlar os seguintes elementos: a quantidade de pedestres esperando para cruzar, utilizando sensores de movimento acoplados aos postes; a presença ou não de um carro aguardando a abertura do sinal, a partir de sensores de loop de indução, montados abaixo do asfalto durante a construção da rua; a velocidade de quantidade de veículos se aproximando do sinal, utilizando sensores de velocidade colocados vários metros antes do cruzamento. Além disso, o sistema ainda comunicaria com outros semáforos próximos a ele e tratar as informações recebidas, modificando ou não a troca das luzes do sinal e manter e um registro histórico do estado do trânsito como forma de previsão de possíveis impeditivos

---

[6] Hypertext Transfer Protocol - https://developer.mozilla.org/en-US/docs/Web/HTTP

[7] MQ Telemetry Transport - http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html

no trânsito. A Figura 1 a seguir demonstra um modelo de cruzamento com os sensores descritos acima.

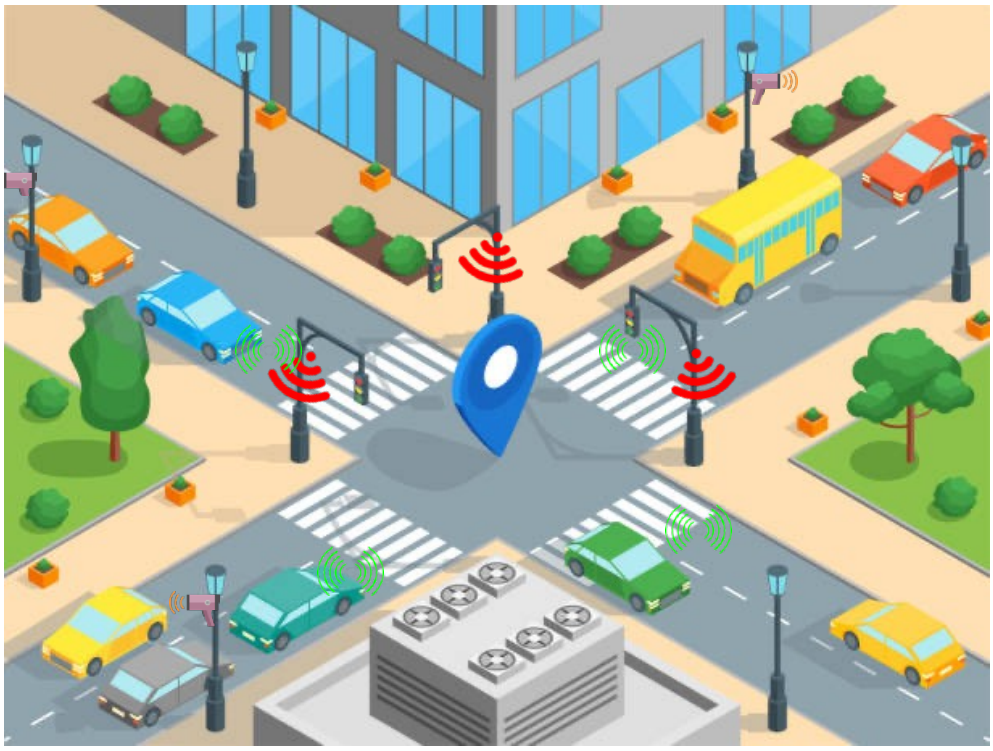


Figura 1 - Modelo de cruzamento e a localização de cada um dos sensores: vermelho sendo os de presença, verdes os loops de indução e os roxos com laranja sendo os sensores de velocidade.

Um sistema como o proposto por esse trabalho requer vários componentes comunicando entre si cada um com suas dificuldades de processamento, de espaço em disco, de comunicabilidade e com isso, é preciso uma boa definição da estrutura como um todo.

Apesar de saber da existência de uma área chamada teoria de fluxo de tráfico, o escopo do trabalho é apenas utilizar conceitos base e teorias nesse domínio. O objetivo não é inovar na área de estudo de tráfego ou sistemas de controle, e sim um estudo sobre o problema, possíveis arquiteturas de tais sistemas e um estudo sobre a escalabilidade da solução desenvolvida e a viabilidade de ser implementado nas vias do Rio de Janeiro. O projeto também não se propõe em implementar esse sistema em ambientes reais, apenas em simulações utilizadas para confirmar o cumprimento dos requisitos e seu bom funcionamento.

IV. Atividades Realizadas

Antes da começo do desenvolvimento do projeto, sempre houve um interesse meu em cima de sistemas IoT ciber-físicos, (sistemas que integram elementos computacionais com elementos físicos), e algumas experiências com

projetos desse tipo, A diferença desse projeto para os outros é que esse existe muito mais um estudo e análise da arquitetura do sistema, além de não se implementar nada em algum desses microcontroladores, Vendo provavelmente a necessidade de se fazer um simulador gráfico, como eu já havia feito algumas aulas durante o curso, decidi utilizar o framework Löve2D[8], que será explicado mais a frente, o que facilitou significativamente seu desenvolvimento. Como já fora mencionado, o projeto também serviu como aprendizado para novas ferramentas, como o Redis[9], que nunca tinha estudado ou aplicado em nenhum projeto anteriormente.

O primeiro passo realizado, após a identificação do problema a ser resolvido; dos requisitos de tal sistema e também dada a arquitetura que desejava-se implementar (quais informações, com que taxa devem ser enviadas), era a escolha de que plataforma serviria de base para o trabalho. A princípio, haviam duas possibilidades: uma utilizando algum *middleware* de comunicação com componentes *IoT*, ou montando a própria estrutura de comunicação, armazenamento e processamento mantendo em vista as condições do problema. Um exemplo de *framework* seria o *Fiware*, uma plataforma open-source que providencia a possibilidade de sensores *IoT* com gerenciamento de informações de contexto histórico de forma fácil por sua excelente reputação em projetos *Smart City.* [13] Após uma análise do sobre o essa estrutura, foi possível concluir que os benefícios desse sistema eram a segurança e a confiabilidade, porém pecava muito na sua alta complexidade e possivelmente podendo causar muito *overhead,* aumentando desnecessáriamente toda a arquitetura do sistema.

Outra possibilidade de arquitetura estudada para o projeto foi uma onde não se utilizaria nenhum *framework* específico. Nesse caso, seria preciso desenvolver cada um dos componentes e como interagiriam e garantir a segurança e confiabilidade em cada um dos estágios. Uma das opções seria a utilização do protocolo *MQTT* entre os componentes do sistema. Um protocolo como o MQTT seria utilizado, devido ao seu baixo custo computacional e conceito de *publish* e *subscribe*, bastante útil para casos de envio constante de informações de sensores, e também responsável pelo processamento desses dados e atuação na luz do sinal. Os benefícios de tal modelo eram que era possível, com o passar do projeto, moldar cada um dos módulos para caber nas necessidades e empecilhos durante o caminho, perdendo com isso a praticidade e possível confiabilidade e segurança que um *framework* daria ao trabalho.

---

[8] Framework de pacotes gráficos para Lua - https://love2d.org

[9] Banco NoSQL em memória - https://redis.io

Comparando ambas as possibilidades, a opção de montar a arquitetura sem uso de nenhum *framework* foi a desejada por razões de ser mais maleável durante o processo todo, de lidar com diferentes tecnologias que considerava interessante e por oferecer uma oportunidade de um estudo mais aprofundado em cada um dos módulos resolvendo seus requisitos dentre as suas restrições.

A atividade realizada depois foi a definição de qual tecnologia seria utilizada para o armazenamento das informações históricas e a comunicação entre os componentes. A princípio, existiam três opções: um banco relacional, como o PostgreSQL[10], um banco não relacional, como o MongoDB[11], e um banco em memória baseado em dicionário chave/valor como o Redis.

O primeiro a ser descartado foi o banco relacional, devido ao fato da estrutura dos objetos de configuração e de log que acabariam por adicionar um nível de complexidade desnecessário ao trabalho, necessitando de joins e queries mais complexas além de um estudo mais aprofundado de como as tabelas seriam desenvolvidas.

A segunda possibilidade, a mais adequada antes do início da investigação, era a utilização de um banco não relacional, se beneficiando da consulta de documentos não estruturados, como os objetos de configuração descritos acima, para manter os dados dos cruzamentos e de histórico no banco de forma organizado e de fácil acesso. Nessa visão, cada interseção seria encarregada de fazer inserções ao banco a um intervalo constante de segundos, por exemplo 5, atualizando seu estado atual e seus valores de sensores. Com essas consultas, seria possível ter uma visão histórica de todos os sensores e mudanças durante o dia. Além disso, seriam feitas consultas de tempo médio dado um intervalo de tempo, para se chegar aos componentes de intervalo histórico que entram na conta da programação de abertura e fechamento do sinal. Uma consulta como essa pode ser realizada com facilidade no Mongo utilizando uma *query* simples. O problema surge, porém, na necessidade de comunicação entre as interseções vizinhas. É necessário que as interseções estejam em constante transmissão de dados por questões de sincronismo entre todos. A interseção seria obrigada a fazer múltiplas consultas, além de ficar extremamente obscura uma comunicação entre as interseções via consultas em um banco. Com isso teria que ser aplicado algum processo de troca de mensagens, que adicionaria ainda mais complexidade ao trabalho.

A última opção sob análise foi a utilização de um banco em memória, no nesse caso o Redis, O Redis é plataforma nova de banco em memória com várias outras ferramentas embutidas. Essa análise foi a mais promissora pois

---

[10] SGBD para SQL - https://www.postgresql.org

[11] SGBD para NoSQL - https://www.mongodb.com

combinava todos os prós de um banco não relacional, sem deixar a desejar nos mesmos pontos. Além de uma política de armazenamento de strings de chave/valor, ele também disponibiliza uma plataforma de troca de mensagens eficaz, e até mesmo possibilidade de utilizar o conceito de *publish* e *subscribe*, combinando todos os requisitos desse sistema cuja arquitetura desejamos definir. Como incentivo, ainda dava a oportunidade de trabalhar com uma nova tecnologia, cujo uso tem crescido consideravelmente, e aplicá-la para um exemplo na vida real.

Dentro da escolha do Redis, foi necessário fazer alguns testes pequenos para garantir o funcionamento dentro do esperado. Um desses testes foi utilizar o Redis em um pequeno esboço do projeto, onde foram desenvolvidos 3 processos, o do agregador, o do gerenciador e o da interseção. Cada um deles faz os mesmos comandos GET/SET/PUSH/POP que seriam usados na arquitetura, porém passando apenas strings pequenas. Também foi colocada uma lógica pequena de agregação e de impressão na tela pelo gerenciador garantindo a chegada das informações esperadas. Com esse pequeno teste foi possível garantir que o modelo funciona e que o Redis é fácil de se utilizar, como esperado.

Já o segundo teste que foi feito, era composto de um visualizador de nós que serviriam de interseções em um modelo real, utilizando a tecnologia Love2D. A Figura 2 mostra uma imagem dessa interface. Na inicialização do programa, é possível arrastar setas entre os nós, identificando eles como vizinhos e também a direção do tráfego entre eles. Após a configuração, o programa criava instâncias de interseções, que se comunicavam pelo Redis, servindo como uma simulação básica da arquitetura do programa. Com o mouse, ao apertar um dos sinais, criava-se uma troca de sinais no nó escolhido e tal troca se propagava em tempos pré-definidos para nós vizinhos. Esse teste serviu como um pequeno protótipo do funcionamento do sincronismo e de criação de configuração das interseções. Ambos os testes foram desenvolvidos utilizando a linguagem Lua.

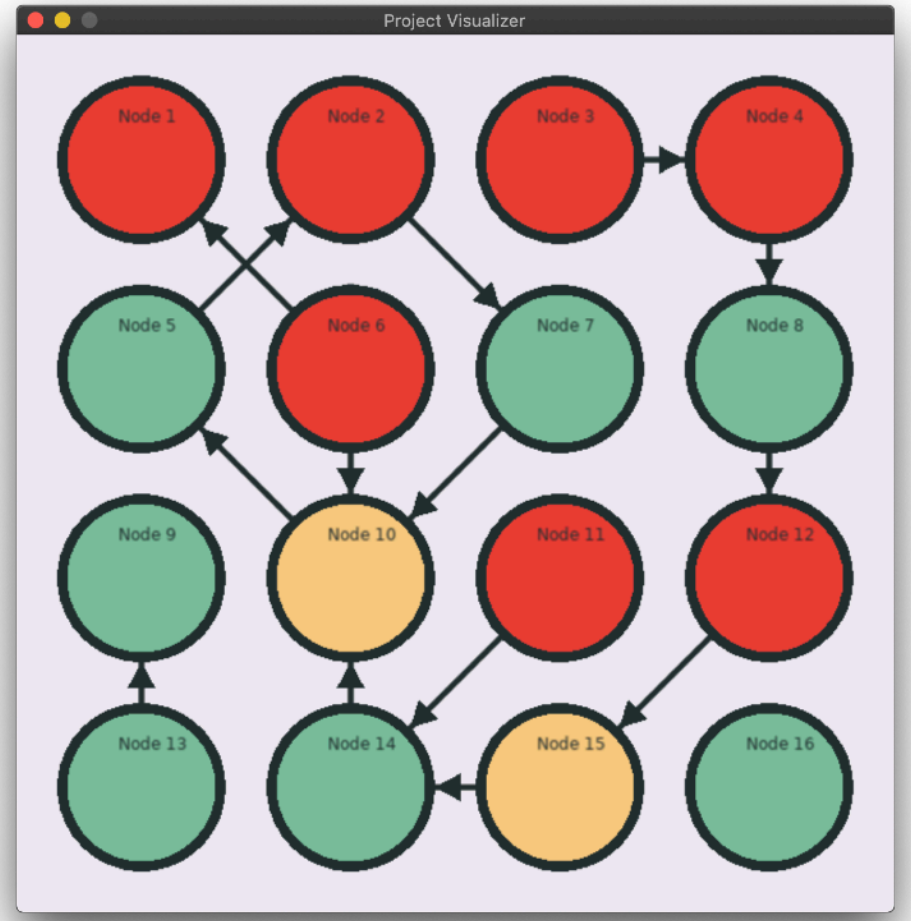


Figura 2- Interface do visualizador desenvolvido para testes

O próximo passo a ser desenvolvido no projeto foi implementar um ambiente de simulação que pudesse agir como uma pequena prova de conceito da arquitetura e de seus benefícios e também como forma de *feedback* do sistema para que pudesse ser melhorado iterativamente com simulações.

A ideia, a principio, seria utilizar dados de sensores e tráfegos reais para que a comparação do antes e depois do sistema fosse mais ligado com a realidade de uma área. Com isso, fizemos uma pesquisa de quaisquer projetos, sistemas ou tabelas que pudessem ser utilizadas para essa finalidade. Encontramos inúmeras oportunidades como APIs em tempo real com quantidade e média de velocidade de carros para cidades do interior da Itália [14], tabelas com fluxo médio de carros por hora em Chicago [15], entre alguns outros repositórios de informação. Contudo, nenhum continha todas as informações que seriam necessárias pra gerar uma simulação completa, como número de carros parados no sinal, tempo de abertura e sincronismo entre as interseções, número de carros trafegando pela rua, número de pedestres atravessando. Além do mais, a grande maioria reportava os dados apenas em um intervalo muito

grande, como 10 minutos, 1 hora ou até mesmo médias diárias, o que não são especialmente úteis para a implementação do projeto pois necessitava-se muitos valores para que se pudesse fazer uma comparação do antes de depois do sistema.

Com isso, a tarefa do Simulador se apresentou como uma tarefa que, dado o escopo que se imaginava, demandaria muito trabalho e poderia ser um projeto final de curso por si próprio. O simulador precisaria tratar de várias interseções, os efeitos de cada uma sob o trânsito, mudança do fluxo dependendo do horário, a visualização de todo esse sistema, assim como também simular de forma leal o comportamento de carros, impossibilitando que fosse desenvolvido por completo. Logo, ficou resolvido que a simulação trataria de uma parte pequena porém bem importante da nossa arquitetura, que seria o sincronismo entre as interseções e partes do sistema que lidam com o cálculo do intervalo de abertura, mesmo dentro de uma visão microscópica, e que seriam, possivelmente, desenvolvidos outros testes para as outras áreas do projeto. Desenvolvemos duas versões do simulador, a segunda com uma simplificação dos carros e com uma quantidade maior deles. Na primeira simulação eram utilizados *sprites* de carros, em comparação a apenas círculos na segunda, requerendo uma lógica maior para seu funcionamento pois a carros são mais longos que compridos.

Seguindo com a plataforma de desenvolvimento de outros testes, desenvolvemos um sistema de simulação utilizando a framework Löve2D com base em Lua, pela sua facilidade na montagem de ambientes parecidos como esse e também pela experiência prévia com essas tecnologias em outras aulas cursadas dentro da universidade.

A primeira versão do simulador, consistia em duas faixas de mão única na horizontal que é cruzada por duas ruas, também com duas faixas de mão única na vertical. Na segunda versão, aumentaram o número de faixas para um total de 8 faixas. A cada cruzamento existem dois sinais e duas faixas de pedestres, dependendo do fluxo dos carros. Os carros param antes da faixa caso o sinal esteja fechado e também tomam cuidado para não bater com outros carros a sua frente. Os carros, a partir da segunda versão também podem fazer curvas nas interseções.

Essa simulação consiste de uma rua com duas faixas na horizontal que é cruzada de por duas ruas, também com duas faixas na vertical. A cada cruzamento existem dois sinais e duas faixas de pedestres, dependendo do fluxo dos carros. Os carros param antes da faixa caso o sinal esteja fechado e também tomam cuidado para não bater com outros carros a sua frente. Os sinais são montados de forma que eles estão sincronizados, significando que ambos

fecham e abrem juntos, evitando um estresse desnecessário no trânsito. A Figura 3 é uma imagem da primeira simulação em funcionamento. A Figura 4 é uma imagem da segunda simulação em funcionamento.

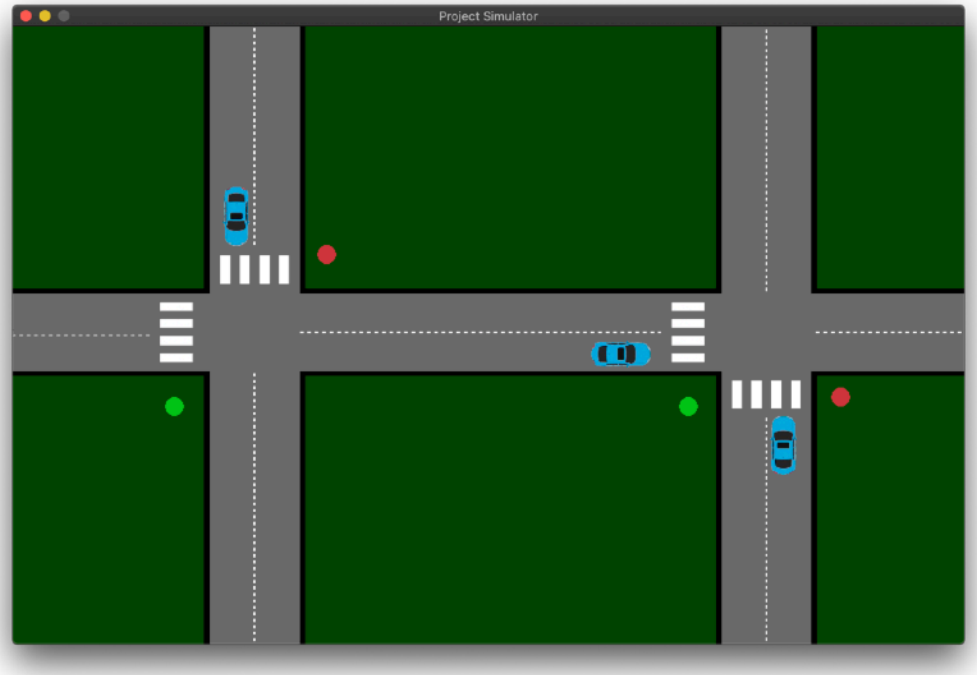


Figura 3 - Interface da primeira simulação desenvolvida para testes

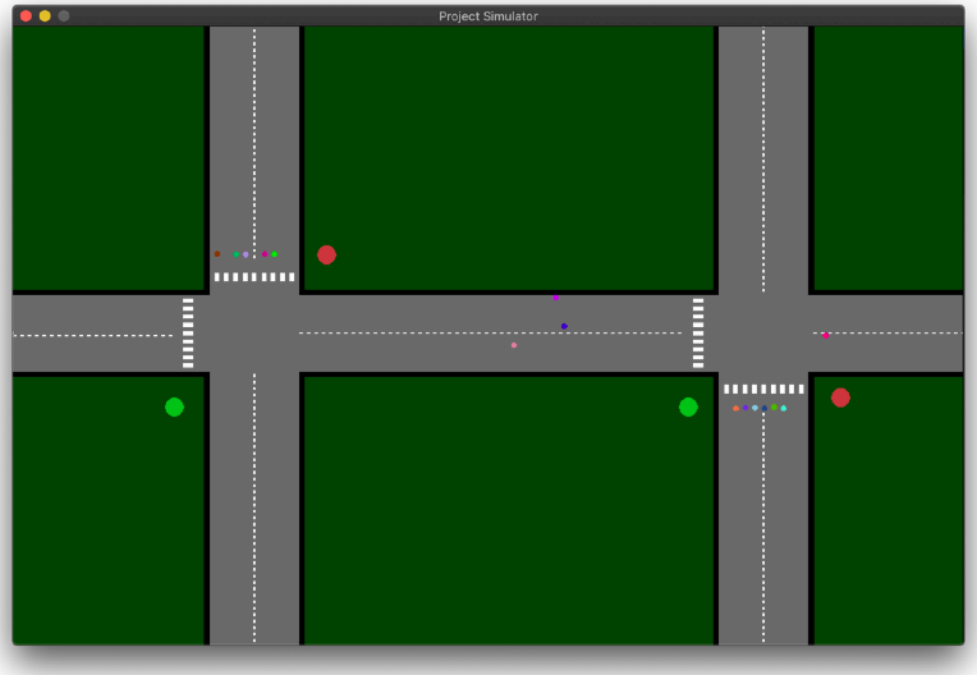


Figura 4 - Interface da segunda simulação desenvolvida para testes

A arquitetura desse sistema se dividiu em quatro partes: o visualizador em si, que chamaremos de Controlador, as Interseções, os Veículos, e o Grid de

Controle de Movimentação. Essas últimas três partes foram desenvolvidas como classes que são utilizadas pelo Controlador que gerencia a simulação como um todo.

O Controlador, que segue os padrões do Löve2D framework, inicializa, no começo do programa todas as variáveis que serão usadas, incluindo a lista de carros, as interseções, e o grid de controle das posições que vão ser explicadas adiante, e depois provê ao framework duas funções que são chamadas a cada ciclo: a função update( ) e draw( ). Dentro da função update( ), o controlador verifica e atualiza a posição dos veículos, inserindo novos aleatoriamente toda vez que um carro sai da tela, sem ultrapassar o máximo definido. No mesmo método, o controlador também verifica se existe a necessidade da mudança de mudança de algum dos sinais e os modifica de acordo. O controlador, que toma conta de uma lista dos objetos que estão na tela, faz esses passos via métodos em cada um dos objetos. Já na função draw( ), utilizando os métodos de desenho na tela do Love, são explicitados o que deve ser visualizado na tela, seja uma imagem como as do carro, ou círculos coloridos, como os sinais. Na primeira versão, os sinais eram controlados apenas por cliques do mouse, mudando para utilizar um protótipo do visualizador na segunda atualização.

Uma das classes mais importantes do simulador é o Grid de Controle de Movimentação, que é responsável por bloquear a passagem dos veículos para os casos de sinal fechado ou de outro veículo a sua frente. Para implementarmos esse grid, dividimos a tela em blocos de 10 por 10 pixels, que são representados em uma matriz por valores de *true* ou *false*. A cada movimentação, o veículo, que ocupa um número específico de blocos, verifica o valor do próximo bloco a sua frente, pois depende desse valor para se movimentar. Caso valor correspondente ao bloco seja *false,* carro não se movimenta e aguarda a próxima verificação, caso contrário, ele “ocupa" esse novo bloco da frente, dando valor true a ele na matriz, e "desocupa" o seu último bloco. O mesmo ocorre com os sinais, ao trocarem de cor, eles “ocupam" ou “desocupam" todos os blocos equivalentes a faixa de pedestre, como forma de bloqueio da passagem de novos veículos. Esse grid é essencial para a simulação pois é o que dá a realidade ao programa, com os carros procurando evitar chocarem-se ou avançarem sinais vermelhos.

Utilizando a mesma lógica do grid de controle de movimentação, foi implementado também um grid de possibilidade de se fazer a curva. Na inicialização das interseções, que serão discutidas a seguir, é declarada uma área de blocos onde, caso o carro passe por esse blocos, ele pode fazer uma curva. O valor guardado no grid é a rotação que o veículo terá caso faça a curva.

Com isso, existe uma lógica de aleatoriedade para as curvas para que apenas alguns carros vire para a outra rua.

Já componente mais simples é o do veículo, cujo objetivo é apenas andar para o bloco a seguir, tomando em consideração a ocupação daquele bloco. Na primeira versão o veículo ocupava 7 blocos ao longo do comprimento do carro, devido ao tamanho da *sprite* do veículo. Já para a versão seguinte, eles ocupam apenas 1 bloco, facilitando muito na hora de fazer as curvas, pois não existe necessidade de ter que olhar para posições atrás no grid por causa da parte traseira do veículo. Essa mudança possibilitou, de forma consistente, que os carros pudessem trocar de pista durante a simulação.

O último componente é o da Interseção. Ela controla a cor dos sinais na interseção e o sincronismo entre ambos. Ela foi desenvolvida de tal forma a gerar um área de atuação, passado como parâmetro no construtor da sua classe, onde ela tem controle da direção do fluxo dos carros. Dentro dessa área, ele controla a faixa de pedestres e também, como mencionado previamente, a possibilidade de curvas dos carros. Outro controle é sobre a cor dos sinais. Primeiramente as interseções se comunicavam entre si via o Redis, executando comandos GET e SET para a verificação da abertura ou não das outras interseções de qual ela estava interessada. Essa comunicação que nos dava o sincronismo. Já na versão seguinte, esse controle passou a ser apenas do valor da cor do sinal que estava sendo gerada pelo sistema. Existe apenas um processo de verificação da cor e bloqueio da faixa pedestres de acordo com a mudança das cores.

O objetivo da simulação a principio era confirmar o bom funcionamento do sistema ao nos possibilitar a medir, utilizando dados reais, o impacto do sistema no número de carros que trafegassem na via e o tempo médio que ficariam no trânsito. Infelizmente, para esse projeto, essa implementação seria inviável e por isso se decidiu em focar em um ponto do sistema e desenvolver um ambiente específico. Apesar da simplicidade da simulação, ainda será possível tirar algumas conclusões e algum aprendizado em cima dele pois podem ser testados tanto o impacto do sincronismo quanto o funcionamento dessa parte do sistema. Outro ponto é que serve como forma de visualização do sistema, ajudando a encontrar possíveis equívocos ou falhas no processo que ao ver em ação podem ser detectados.

O método de desenvolvimento do trabalho foi de se implementar o sistema acoplado a testes de tal forma a que possa haver uma iteratividade de melhorias tanto no sistema quanto nos testes sendo feitos. Ou seja, ao implementar novas partes do sistema, foram desenvolvidos pequenos testes de

conceito garantindo o funcionamento e também melhorando seu desempenho através da análise do resultado dos testes.

A seguir, seguem os cronogramas do desenvolvimento do projeto. O primeiro sendo o que foi projetado no final do Projeto Final I e o segundo que reflete o que o foi feito durante os meses. Como explicado anteriormente, se gastou muito tempo no desenvolvimento do simulador e com isso ouve um atraso significativo em todo projeto, causando com que a últimas semanas de Novembro fossem a com maior trabalho.

Projetado

Agosto	Setembro	Outubro	Novembro	Dezembro
Desenvolvimento da Simulação				
	Implementação do Sistema			
		Projeto, desenvolvimento e análise de testes		
			Desenvolvimento do Relatorio de Projeto Final	

Atual

Agosto	Setembro	Outubro	Novembro	Dezembro
Desenvolvimento da Simulação				
		Implementação do Sistema		
			Projeto, desenvolvimento e análise de testes	
			Desenvolvimento do Relatorio de Projeto Final	

V. Projeto e Especificação do Sistema

i. Panorama da arquitetura

Partindo da ideia que era necessário que houvesse um componente em cada interseção, responsável por apenas aquele cruzamento, fazendo as verificações dos sensores e atuando sobre a sinalização, a arquitetura se baseou primeiro na necessidade de que existam várias interseções, algumas delas que se comunicam entre si por razões de sincronismo do tempo de cada sinal. O sistema, porém, também necessitaria: providenciar uma visualização em tempo real do estado de cada sinal; disponibilizar uma forma de sobrepor o tempo de sinalização utilizado naquele momento para uma interseção, uma espécie de "ordem" para o sinal; e uma gravação do histórico de cada estado para que haja a possibilidade de um estudo e aprimoramento do algoritmo de cálculo do tempo. Dadas essas necessidades, se optou por introduzir um novo componente ao sistema, o Gerenciador, cujo papel é deixar que o usuário do sistema possa enviar “comandos" para a interseção, servindo também como ponto de acoplamento para um possível um módulo visualizador, pois ele receberá as informações em tempo real de cada interseção, possibilitando também que apenas um componente se comunique com o banco de armazenamento desses resultados, o banco de log. O gerenciador também é encarregado de agregar os tempos de abertura e fechamento do sinal para uso da interseção no cálculo do intervalo entre cada mudança, introduzindo um aspecto histórico para a interseção. A Figura 5 exibe a arquitetura descrita.

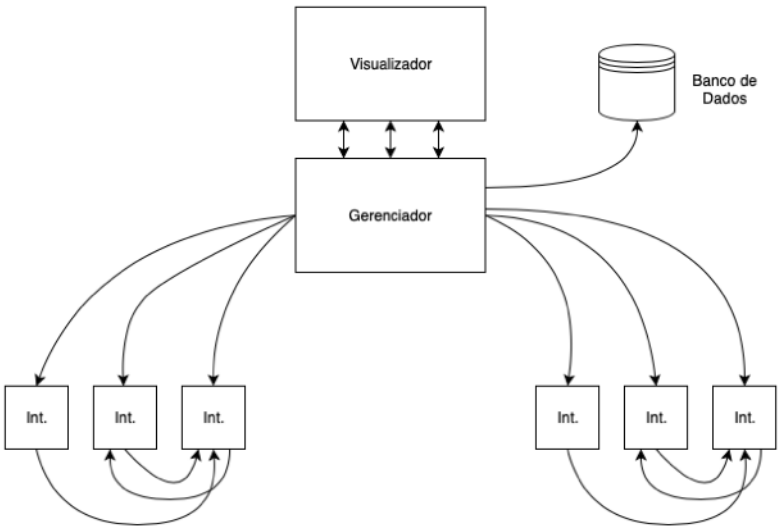


Figura 5 - Visão panorâmica da arquitetura do projeto

Nota-se que a comunicação em tempo real ocorre apenas de duas maneiras: entre o Gerenciador e as interseções, ou entre uma interseção e outra interseção. O Gerenciador recebe de cada interseção um objeto de log, contendo informações de qual estado do sinal naquele momento, quantas pessoas apareceram para atravessar, quantos carros estão parados no sinal, e quantos carros passaram pelo sinal, dado um intervalo de tempo também incluso no objeto. Essas informações são requisitadas a cada 5 segundos pelo o Gerenciador. O Gerenciador também tem controle de quais interseções estão operantes, para um controle melhor das requisições. O Gerenciador também, quando configurado pelo usuário, pode mandar para uma interseção específica qual deveria ser seu intervalo entre cada cor do sinal (em ocasiões especiais como eventos).

Entre as interseções, existe a passagem de informações de estado atual e intervalo entre abertura e fechamento, que podem entrar no cálculo, e também de quando será a próxima troca de sinalização, para que haja sincronismo entre cada um dos cruzamentos. Essas comunicações ocorrerão em intervalos de 1 seg.

Como mencionado anteriormente, para implementar essa arquitetura, foi escolhido o Redis por ser capaz de providenciar ambas as comunicações em uma tecnologia comum, pois existem bibliotecas de comunicação com o Redis para vários microcontroladores, além do alto desempenho por se tratar de banco em memória, possibilitando uma escalabilidade maior do sistema como um todo.

Essa arquitetura é escalável pois com o aumento no número de interseções, a comunicação entre elas continua a mesma, sem que haja um acréscimo no número de vizinhos, e o Gerenciador pode ser dividido em áreas distintas de uma cidade, de forma a não sobrecarregá-lo, deixando com que o Visualizador possa ainda desempenhar seu papel se acoplando a todos os Gerenciadores de cada área.

ii. Utilização do Redis

Após um estudo das melhores opções para as características necessárias, o Redis, plataforma nova de banco em memória com várias outras ferramentas embutidas, foi a mais promissora pois combinava todos os prós de um banco não relacional, sem deixar a desejar nos mesmos pontos. Além de uma política de armazenamento de strings de chave/valor, ele também disponibiliza uma plataforma de troca de mensagens eficaz, combinando todos os requisitos desse sistema cuja arquitetura desejamos definir. Como incentivo, ainda dava a oportunidade de trabalhar com uma nova tecnologia, cujo uso tem

crescido consideravelmente, e aplicá-la para um exemplo na vida real. Definido Redis como escolha, foi desenvolvida a seguinte estrutura de comunicação com o banco, como demonstrado na Figura 6.

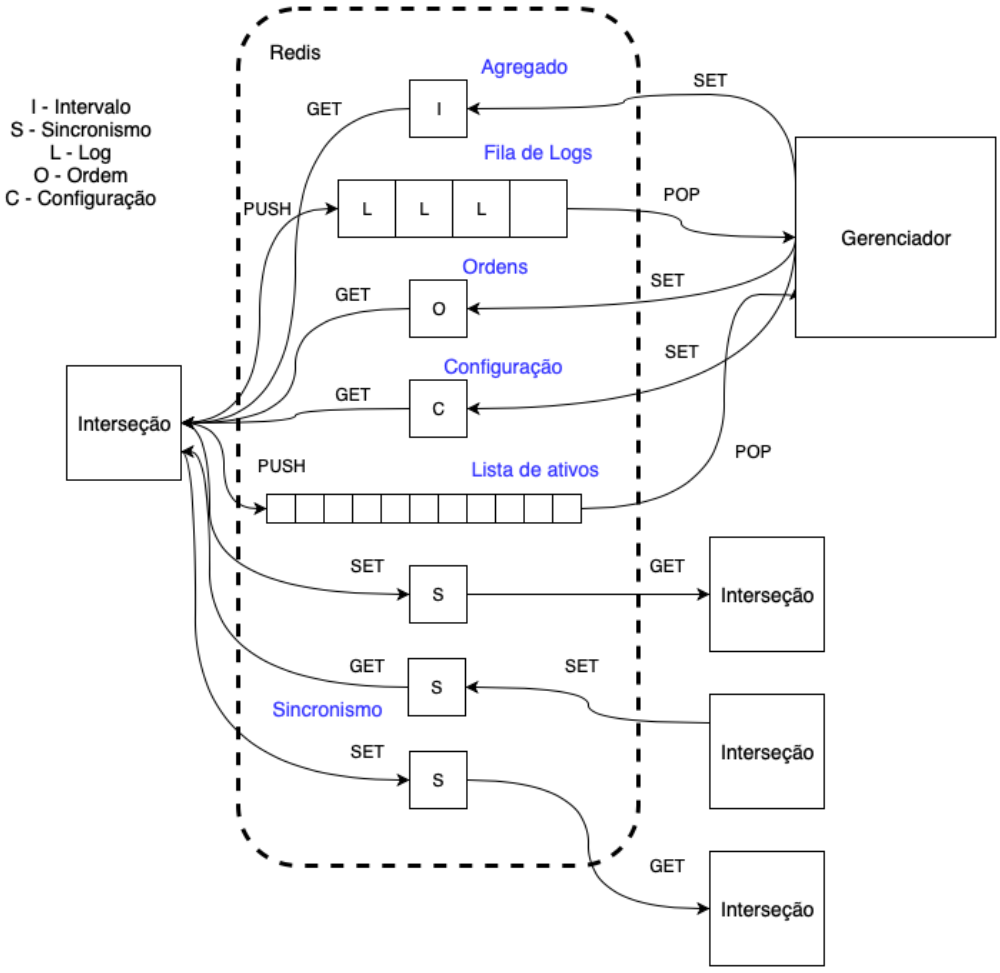


Figura 6- Arquitetura de comunicação dos componentes utilizando o Redis

Nessa arquitetura, temos que a interseção faz toda sua comunicação pela plataforma do Redis, que oferece confiabilidade e alto desempenho [16]. Apesar de muitos desses campos pudessem ser atualizados utilizando o conceito de *publish* e *subscribe*, infelizmente, esse aspecto do Redis não é presente nas bibliotecas para microcontroladores como o redis-lua[12], ou existem limitações de não se poder fazer outras chamadas não pub/sub como redis-py[13] fazendo com que tal opção não fosse viável para esse projeto.

Observando a Figura 6, destacada anteriormente, nota-se a grande quantidade de diferentes objetos que são enviados para o banco, começando pelo valor de agregação, gerado pelo agregador de dados históricos dentro do

[12] https://github.com/nrk/redis-lua

[13] https://github.com/andymccurdy/redis-py

gerenciador, que será explicado nas próximas seções. Essa troca de informações será feita via uma operação SET em uma chave, ou seja, dando uma chave no banco um valor específico, e uma operação GET de tal chave, que retorna o valor associado a aquela chave. Dessa forma, esse valor pode ser atualizado a qualquer momento pelo agregador e não há a necessidade de se guardar objetos passados. O valor guardado é um objeto de intervalo que contém valores agregados de dias passados de intervalos de tempo entre cada troca de estado, e até quando esses intervalos devem ser usados no cálculo interno do componente. Esse comando ocorre apenas a cada 10 minutos.

Também foi projetada uma fila estilo FIFO (First In First Out) para que as interseções possam incluir na fila seus objetos de visão histórica, os logs, Esses objetos serão explicados nas seções seguintes, porém é necessária que sejam recebidos todos esses objetos, para que possam ser inseridos no banco de dados em disco com todo o histórico e com isso haja uma visão completa de toda a evolução dos dados. Será incluído um novo objeto na fila a cada 5 segundos.

Haverá também a utilização de par chave valor para as ordens que serão enviadas para cada interseção. A interseção, periodicamente vai requisitar o valor da chave para verificar se existe alguma ordem que deve ser tomada e se sobrepor ao cálculo. Esse comando ocorre de 2 em 2 minutos.

Para que haja uma consistências nas chamadas ao Redis, evitando chamadas desnecessárias a interseções que estão fora do ar por alguma razão, é necessário um controle das interseções ativas. Com isso, existe uma lista de interseções ativas, que periodicamente, cada cruzamento deve inserir seu identificador para que o gerenciador possa consumir essa lista e com isso saber de quais chaves ele devem requisitar. Nos casos de interseções que voltaram a atividade recentemente, é colocado dentro do banco um objeto de configuração, que também será discutido nas próximas partes. A interseção fará essa inclusão a cada 10 minutos.

Já em relação a comunicação entre as interseções, será armazenado, por cada interseção, um objeto de sincronismo, que vai servir como forma de ajustar os tempos de abertura e fechamento do sinais para serem em tempos próximos, diminuindo o tempo de travessia pela via. Esse objeto consistirá de dados sobre o intervalo de tempo dedicado a cada estado do sinal, estado atual da interseção, assim como o tempo da última troca e a próxima troca esperada. Um exemplo desse objeto está descrito no Apêndice 1 em formato JSON[14]. A cada 1 segundo, as interseções vão se comunicar com os seus vizinhos.

[14] JavaScript Object Notation

Como essa estrutura vai estar toda no Redis, é necessário que hajam proteções para evitar possíveis ataques maliciosos que afetem todo o trânsito da cidade. Para isso existem modos de proteção dentro do Redis que permitem acesso de apenas endereços especificados, dificultando o acesso de infratores. Apesar de não haver controle de acesso ao Redis, é possível configurar para que haja uma chave de autenticação de acesso, obrigando o cliente a se autenticar antes de qualquer consulta.

iii. Descrição dos módulos

a. Módulo da Interseção

O módulo da interseção é o componente que estará fisicamente presente em cada um dos cruzamentos da cidade, e será encarregado de fazer toda a manipulação dos dados de sensores, fatores históricos e nós vizinhos para gerar um intervalo de tempo a minimizar todos os problemas já destacados. Cada interseção terá uma configuração que detalhará os sensores (quantos estão presentes e quais) e o arranjo das vias (se são de mão dupla, quais direções são possíveis) que estará armazenado no banco que está acoplado ao Gerenciador, que repassará esses dados para a interseção.

Esse objeto de configuração vai conter um campo com o identificador da interseção — basicamente um UUID —, uma lista dos identificadores das interseções consideradas vizinhas, e uma descrição das vias divididas em direções cardeais. Para cada ponto cardeal, pode existir duas direções, vindo em encontro do centro do cruzamento ou saindo do centro. Dentro de cada uma dessas direções, existe a descrição dos sensores presentes — identificadores, e tipos—, se há ou não uma sinalização naquela direção, quantas pistas existem e se existe uma faixa para pedestres atravessarem. Um exemplo desse objeto está descrita no Apêndice 2 no formato JSON.

Outro ponto de comunicação com Gerenciador é o processo de envio da visão da interseção em um dado momento, que será utilizado para agregações e armazenamento do histórico. Para essas informações, a interseção enviará a maior quantidade de informações possíveis, para que, posteriormente, se possa estudar todos esses dados e melhorar o modelo como um todo. Com isso, o objeto de log, inclui: estado atual do sinal (vermelho, amarelo ou verde), o intervalo entre cada um dos estados, uma estimativa dos pedestres que estão esperando para atravessar, quantidade de carros parados esperando para avançarem, quantidade de carros que passaram pelo sensor de velocidade, um *timestamp* da última troca de sinal, e a data de envio das informações. Um exemplo, em notação JSON, dessa classe se encontra no Apêndice 3.

Passando pelos passos que o código da interseção vai realizar a cada ciclo: leituras dos sensores, fazendo os cálculos necessários para trazer as informações de pedestres e carros e agregando essas informações, além de verificar se, dado o intervalo calculado, deve se trocar de estado na sinalização. Os valores que serão considerados nesse cálculo serão os dados de quantidade de pedestres a espera, quantidade de carros parados e também que número de carros passaram pela via, dados de sincronismo de interseções adjacentes, valor agregado histórico dado o dia da semana e horário, e se existe alguma ordem do gerenciador que deve sobrepor tal cálculo. A cada 1 segundos, será enviado o objeto de sincronismo, explicado na seção do Redis, para ser consumido pelas outras interseções, assim como será consultado os objetos de sincronismo dos cruzamentos vizinhos. A cada 5 segundos então, será enviado o objeto de log para o gerenciador consumi-lo nos seus processos.

Um ponto vital no funcionamento desse sistema é o sincronismo entre interseções vizinhas que deve ser a prioridade para cada um desses componentes. Para solucionar essa questão, o objeto de comunicação entre os dois, já descrito na seção do Redis, contará com dois campos, um com um *timestamp* de quando houve a última troca de sinal, e outro com quando está programado para se ter a próxima mudança. Com essa informação é possível, dentro da conta realizada previamente, adaptar o intervalo de forma a sincronizar as mudanças, evitando, aqueles incômodos casos onde, trafegando por uma via, se para em todos os sinais pois quando um abre, o próximo acaba se fechando.

b. Módulo Gerenciador

Como previamente já destacado, o módulo do gerenciador, servirá como intermediário para os componentes. Ele será responsável por: se comunicar com o banco de dados estáticos, repassando objetos de configuração para interseções, e salvando os históricos recebidos pelos mesmos no banco; possibilitar o *override* de decisões das interseções, configurando manualmente o tempo de abertura e fechamento dos sinais; agregador dos dados históricos para ser utilizado no cálculo das interseções; e repassar as informações que estão vindo em tempo real para um possível módulo visualizador.

Começando pela comunicação com o banco de dados estáticos e armazenamento dos logs de cada interseção. Para cada interseção, o gerenciador consultará sua configuração no banco de dados, e passará, pelo Redis, esse objeto para quem o requisitou. O bom de se manter tal configuração fora do Redis é que reduz o espaço consumido assim como também facilita na

atualização desses dados posteriormente, caso tenha sido adicionado, por exemplo, um novo sensor na interseção.

A cada 5 segundos, as interseções também mandarão um objeto com informações do seu estado atual para o gerenciador, que por sua vez, armazenará no banco de dados, para ser guardado como log histórico. Esse objeto também fará parte de uma agregação dos dados históricos vistos recentemente, para ser utilizado pela interseção.

Por receber todas essas informações, o gerenciador tem uma visão geral de todas as interseções ativas, que pode ser utilizada por um visualizador que manipula esses dados, antes de serem inseridos no log, para mostrá-los em uma interface gráfica, simplificando para algum usuário que queira consumir esses dados. Uma outra possibilidade do gerenciador, é também possibilitar a visualização de dados históricos, utilizando consultas em cima dos logs no banco de dados.

Outra necessidade aplicada no gerenciador é o envio de intervalos que se sobreponham aos que estão sendo gerados pela interseção, úteis para casos de shows ou de acidentes onde se pretendem desviar o trânsito de forma específica na área, priorizando certas vias por deixarem os sinais contidos nelas abertos por tempos maiores do que o habitual. Isso seria implementado na forma de inserção de objetos de ordem no Redis, que seria, periodicamente conferido pela interseção. Esse objeto descreveria qual intervalo de tempo de cada cor, e quando se “expira”, ou seja, quando deve voltar ao tempo normal.

iv. Recursos Físicos

Para esse projeto, porém, não basta apenas propor a arquitetura sem que também se avalie a questão dos dispositivos físicos e a viabilidade desse sistema em uma cidade como o Rio de Janeiro.

Esse sistema foi projetado para que existam microcontroladores presentes fisicamente em cada uma das interseções, conectados por fios a todos os sensores e via um sinal de rede telefônica, 4G, com a internet. A necessidade de se existir um micro de processamento em cada cruzamento se dá pelo motivo da necessidade de tal sistema funcionar de forma isolada do mundo da internet, possibilitando que o sinal ainda funcione corretamente mesmo sem conexão ao gerenciador. Outra questão é que se não fosse dessa maneira, utilizando a nuvem, haveria ainda a necessidade de algum componente que passaria os dados dos sensores para serem utilizados e as mudanças de sinalização repassadas para esse componente também. Nesse caso, esse componente poderia facilmente fazer os usos dos dados localmente, evitando o

problema mencionado acima. Outras escolhas para aumentar a confiabilidade do sistema foram a conexão dos sensores diretamente com o microcontrolador, retirando a necessidade de outra forma de comunicação entre o sensor e o componente da interseção, e o uso da conexão com o sinal 4G, tirando a dependência de sinais WiFi e de cabos de redes para cada cruzamento.

Os sensores utilizados seriam, a principio, três tipos: os sensores de indução por loop, onde é colocada uma espécie de bobina abaixo do asfalto de tal forma que com a parada de um carro sobre essa bobina, o metal afetará o campo magnético, registrando a presença do carro em frente ao sinal; sensores de presença, muito populares e presentes no dia a dia e que seriam utilizados em um arranjo próximo a faixa de pedestres, possibilitando o cálculo de quantas pessoas estão esperando para atravessar; e por fim, sensores radares de velocidade, que serão utilizados para gravar a quantidade de carros que esão trafegando pela via.

Entrando na questão da viabilidade da implementação desse projeto em larga escala no Rio de Janeiro, é necessário que seja analisada a questão financeira. Utilizando-se como base o número de 2300 sinais no Rio de Janeiro [4], e tirando como média que grande parte dos cruzamentos tem apenas 2 sinais, podemos supor que existam perto de 1100 interseções na cidade. Dado que o custo de cerca de 300 reais para o microcontrolador Raspberry Pi, cerca de 400 reais para sensores de presença, 20 mil para o loops de indução, e 20 mil para os radares — preços para uma interseção contendo 10 sensores de presença, 3 loops de indução e 2 radares — totalizando um total de certa de 41 mil reais. Adicionando um custo de 30 mil para a instalação de cada, temos um total de cerca de 78 milhões de reais, uma pequena parte dos quase 31 bilhões de orçamento anual de 2019 recebido pelo município do Rio de Janeiro. [10]

VI. Implementação e avaliação

Como mencionado anteriormente, optou-se, devido ao tempo e a complexidade dos testes necessários para o funcionamento do sistema completo, se implementar uma visão menor do sistema, verificando apenas seu funcionamento e bom desempenho nas áreas mais importantes. Para testar a funcionalidade e a melhoria que esse sistema poderia proporcionar, foram desenvolvidos alguns testes, ou estudos, sobre alguns aspectos do projeto, como o tempo de travessia de um carro em comparação com um sinal sem sincronismo, e o desempenho e robustez da arquitetura utilizada dentro do Redis.

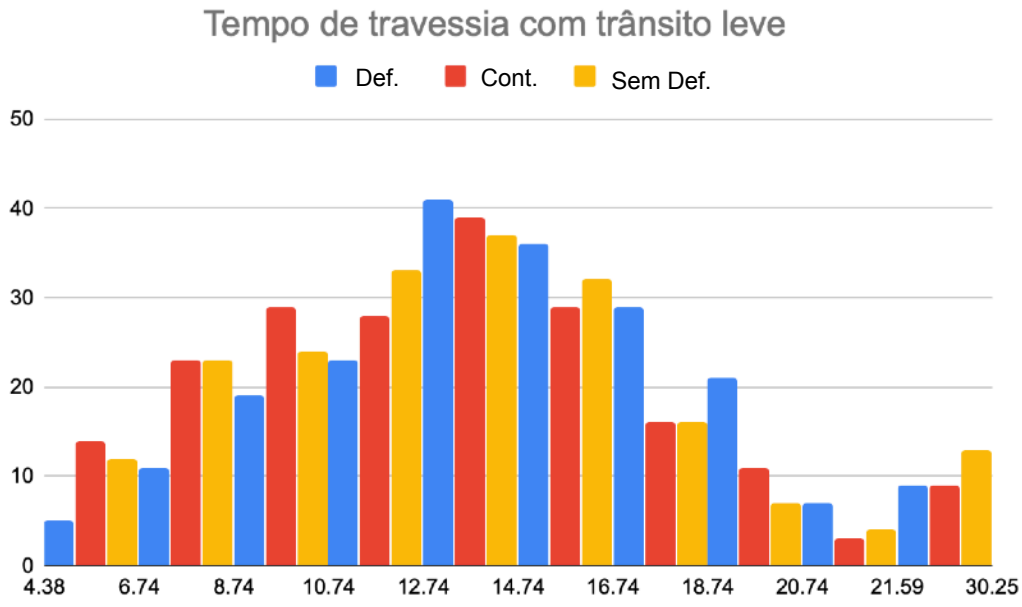
Para podermos verificar se há uma diminuição do tempo que carros demoram para passar por uma via, foi adaptado o simulador tratado

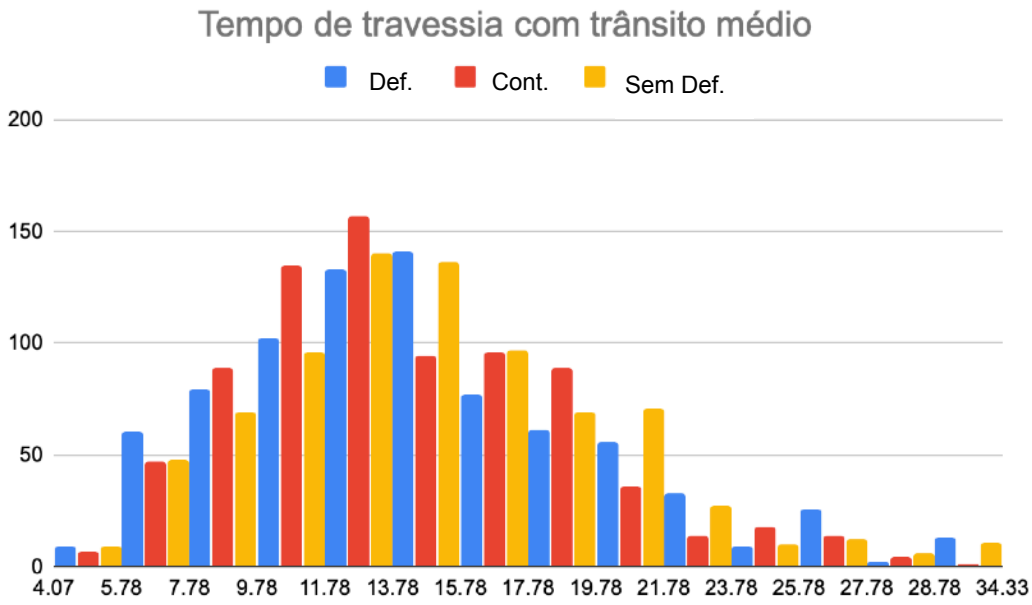
anteriormente para conter mais carros, incluindo a possibilidade de fazer curvas nas interseções, para facilitar o entendimento, os desenhos dos carros foram trocados para círculos, e também diminuídos consideravelmente de tamanho. Abaixo se apresenta uma visão dessa simulação adaptada para esse teste em específico.

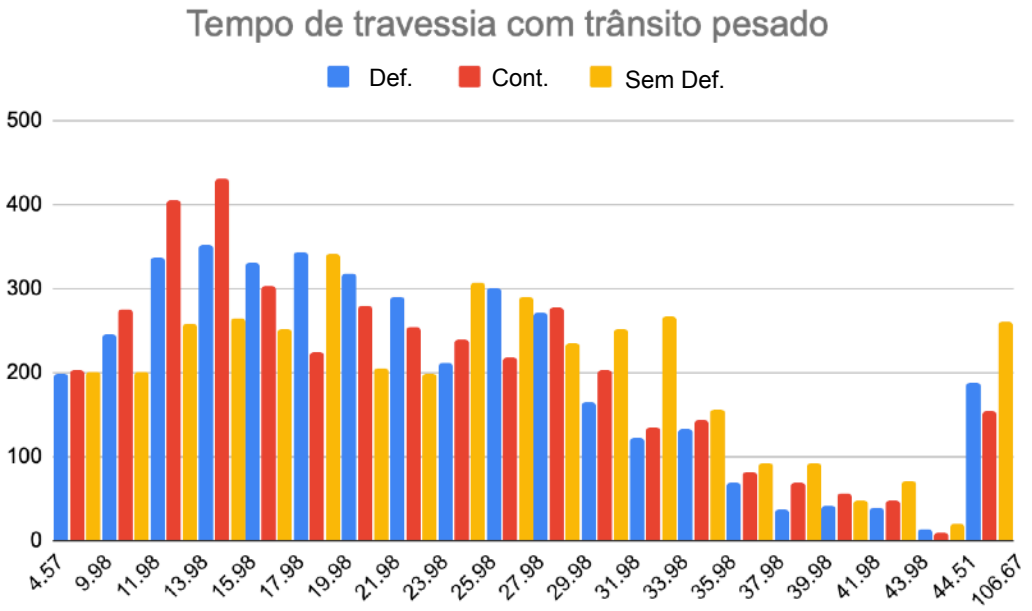
Em relação às interseções, foi desenvolvido também uma versão reduzida do sistema de tal forma que levasse em consideração apenas o timing das interseções, passando entre eles um objeto de sincronização utilizado para se comunicarem e com isso poderem saber quando deve haver uma troca de cor. Existe também uma lógica onde o tempo de abertura do sinal leva em conta a proporção da quantidade de carros em cada faixa e leva em consideração, de forma gradativa, o valor dessa razão. Como comparação, foi medido o tempo que o carro permanece na tela, que podemos considerar como o tempo de travessia dele pela via, assim como a direção dele e se ele fez alguma curva.

Serão comparadas três situações de trânsito: uma que consideramos trânsito leve, com no máximo 50 carros nas vias; trânsito médio, com no máximo 200 carros nas vias; e trânsito pesado, com no máximo 1000 carros nas vias. Queremos mostrar que com o sincronismo do nosso sistema, em comparação com a atualidade, existe uma diminuição no tempo que o carro ficará na via.

Abaixo apresentam-se histogramas para cada situação. Ele mapeia a quantidade de carros e o tempo de travessia de cada um. Dentro de cada caso, foi testado utilizando uma pequena defasagem de tempo entre um sinal e o outro, evitando os casos de bloqueio de cruzamento (Def.) , um com defasagem que também levando em conta o volume dos carros para cada direção (Cont.) , e por fim, um sem defasagem alguma, que abrem e fecham sincronizadamente (Sem Def.)

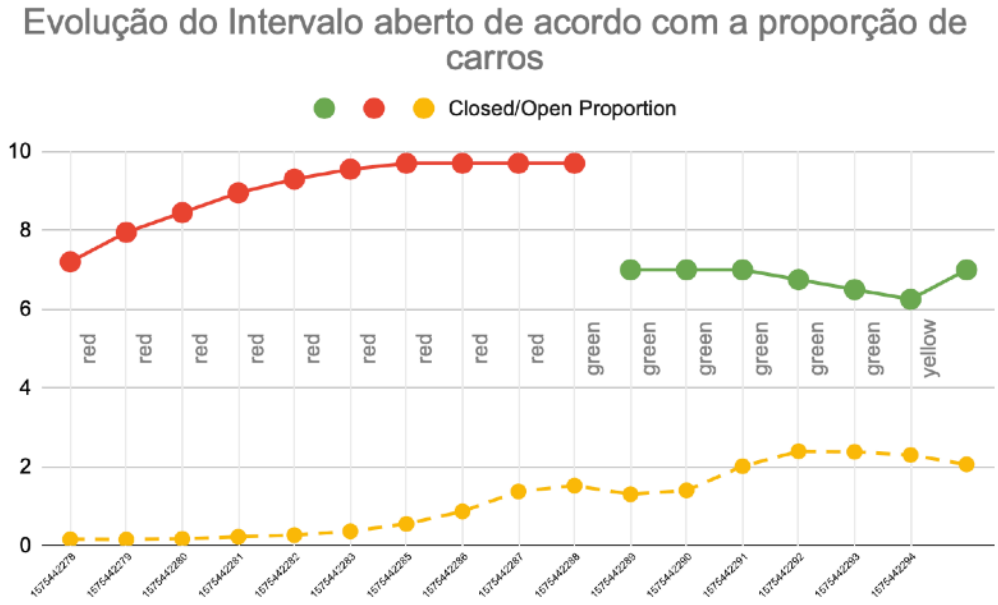






Nota-se que para cada um dos casos, existe uma tendência positiva para a utilização tanto quanto para a defasagem quanto a utilização da quantidade de carros. Por exemplo, no caso de trânsito pesado, notou-se uma melhora de cerca de 15% entre a política de uso de quantidade de carros e a sem nenhum defasagem, apenas sincronismo. Vale ressaltar que encontramos diariamente sinais que não estão sincronizados, ou seja, essa diferença tende a aumentar quando levamos em consideração casos atuais.

Outra coisa interessante a ser observada é como se comporta o intervalo de tempo que o sinal fica naquele estado dado a quantidade de carros. No gráfico a seguir, a proporção em amarelo se relaciona diretamente com quanto tempo fica aberto o sinal, vermelho sendo a quantidade de tempo que pretende-se ficar fechado, e verde quanto tempo ele pretende ficar aberto. Notamos que, dado a proporção pequena entre carros esperando no sinal fechado e passando pelo sinal aberto, a tendência é que o sinal fique fechado para aquele número menor de carro por mais tempo. O mesmo se aplica quando o sinal se abre. Como a proporção de carros esperando para o sinal abrir é maior, o tempo que o sinal fica nesse estado diminui.



Como mencionado anteriormente, o escopo da simulação não nos ajuda a fazer esses testes em ambienteis reais, levando os dados a não serem necessariamente representativos de um cenário real. Contudo, com esses pequenos testes, é possível concluir que, dados cenários parecidos com os apresentados na simulação, qualquer tipo de política de sincronização já pode providenciar diminuições consideráveis no tempo de travessia dos carros por uma via.

Outro teste necessário, é a do desempenho do Redis dentro da arquitetura montada. Como o sistema lida com centenas de interseções, todas se comunicando via um mesmo protocolo, é preciso garantir que dada as piores circunstâncias, o sistema ainda responderá com um tempo não tão elevado a ponto que interfira no seu funcionamento. Dado que o sistema não é muito suscetível a diferenças pequenas de milissegundos, pois os sinais ficam abertos

por mais 30 segundos normalmente, logo uma variação de menos de segundo não afetará o significativamente o tempo de abertura do semáforo, pode-se argumentar que é necessário apenas que o tempo de resposta seja abaixo de 2 segundos em média para que o sistema não seja impedida de desempenhar corretamente.

Para esse teste, foi desenvolvido um pequeno protótipo do sistema, utilizando todas as filas, GET e SET que o sistema faria, passando contudo, strings de tamanho em bytes variável, de 100 bytes a 2kilobytes, tentando se aproximar, e talvez até mesmo extrapolar os objetos que são passados entre cada uma das peças, forçando o máximo possível o Redis a trabalhar. Foram utilizados dois computadores pessoais, um rodando um protótipo do gerenciador e o servidor Redis, e outro rodando um total de N interseções. São medidos os tempos de resposta para cada uma das operações ao Redis.

Para condições de testes mais coerentes com a realidade, foi utilizada duas máquinas Ubuntu utilizando a plataforma de cloud da AWS[15]. Em uma dessas máquinas ficou localizado o servidor do Redis, e em outro o processo do Gerenciador. Ambas as maquinas são do tipo t2.micro, com memória baixa e aspectos de rede padrão dentro da AWS. As interseções foram simuladas no computador fazendo chamadas diretamente ao Redis, que fica na Virgínia, EUA. Devido a um limite de conexões via socket, foi necessário aumentar o número de acessos ao Redis para acontecer 10 vezes mais frequentemente, como forma de simular uma grande movimentação de objetos trafegando pelo Redis sem atingir a barreira física do sistema operacional do computador.

Foram testados com 10, 50 e 250 processos de interseções rodados via um pequeno script em bash. A primeira tabela mostra a média do tempo para cada uma das chamadas ao Redis. Já na segunda, é a mesma métrica porém vista do gerenciador, que está na infraestrutura da AWS.

Número de processos	PUSH to Log	GET agg	PUSH to Active	SET sync	GET sync
10	153.8 ms	139.3 ms	156.2 ms	154.2 ms	154.6 ms
50	169.8 ms	155.5 ms	155.9 ms	163.8 ms	162.9 ms
250	175.3 ms	162.9 ms	163.4ms	176.7 ms	176.5 ms

Número de processos	EMPTY Log Queue	SET agg	EMPTY Active
10	9.16 ms	0.95 ms	5.82 ms
50	9.24 ms	0.97 ms	8.64 ms
250	10.6 ms	0.98 ms	11.6 ms

[15] Amazon Web Services, produto de computação na cloud - https://aws.amazon.com

Apesar do aumento significativo na quantidade de processos fazendo conexões e consultas ao banco do Redis, notamos que não há um aumento significativo em nenhuma das operações principais das interseções. Com isso, notamos que o uso do Redis permite que exista uma quantidade extremamente alta de interseções, ajudando a escalar o sistema como um todo.

VII. Considerações Finais

Existe a necessidade atualmente de atualizar e incorporar a tecnologia nas nossas cidades tornando elas cada vez mais eficientes e melhores para se viver. A inclusão de um sistema de tráfego inteligente seria uma das mais importantes pois afetaria diretamente e diariamente todos da cidade, independente de classe social ou econômica, e como já discutido, não é algo que requer um investimento tão grande.

Com a conclusão do trabalho acredito que a contribuição feita foi no âmbito de acrescentar à discussão a necessidade de acrescentarmos a tecnologia dentro das nossas cidades para que possamos melhorar a qualidade de vida de todos. Avalio que a implementação de um sistema como esse seria extremamente benéfico a sociedade pois poderia diminuir consideravelmente a jornada de trabalho diária da maioria das pessoas, reduzindo o estresse causado pelo trânsito. Apesar do projeto não implementar nada que pode ser aplicado da forma atual, a criação de novos trabalhos e estudos sob esse tópico aumenta o debate e proporciona novas formas de se pensar no assunto.

Todo código desenvolvido para esse projeto pode ser acessado no meu repositório no GitHub pelo link https://github.com/victormeira/projetofinal. O projeto está divido em pastas para cada atividade e teste realizado.

Acredito que esse projeto me proporcionou a oportunidade de estudar inúmeros tópicos diferentes e pensar analiticamente em uma arquitetura a partir do zero e sem respostas consideradas corretas, algo que não é dado na sala de aula e se aprende apenas com a prática e experiência. Pude me aprofundar ainda mais na linguagem Lua, desenvolvida aqui mesmo na PUC-Rio, e o Löve2D pois acredito que possa ser um diferencial no futuro, conhecendo que Lua é utilizada por grandes empresas como Autodesk[16], e Apple[17] e não é comumente dada em outras faculdades do Rio de Janeiro. Também pude me inteirar mais no Redis, extremamente utilizado no mercado e com um nível de abrangência tão grande que pode ser utilizado até mesmo em situações onde não se esperava.

---

[16] http://help.autodesk.com/view/ScaleformStudio/ENU/?guid=__scaleform_studio_help_programming_guide_scripting_lua_html

[17] https://news.ycombinator.com/item?id=17700894

Uma das mudanças que seriam feitas caso o projeto fosse começado agora, seria um foco um pouco mais cedo na simulação dos carros tentando desenvolver algo que se assemelha um pouco mais a uma “rede de sinais”. Acredito que com um simulador mais coerente com a realidade, incluindo pedestres e outras coisas, seria possível uma implementação mais clara e fácil do sistema como um todo. Acredito que ter deixado a simulação para a segunda parte do projeto foi um erro e acabou comprometendo a implementação do sistema como um todo.

Existem inúmeras oportunidades nessa área de controle de tráfego, seja com um sistema que se comunica com os próprios carros, preparando, por assim dizer, para um futuro com veículos anônimos, ou até mesmo uma biblioteca de simulação de tráfego para que novos projetos no assunto pudessem ter mais facilidade na hora dos testes e com isso melhorarem o desempenho do sistema desenvolvido.

VIII. Referências Bibliográficas

[1] - https://oglobo.globo.com/rio/rio-perde-35-bi-com-tempo-gasto-no-transito-pelos-trabalhadores-21243153. O Globo, 2017.

[2] - https://www.atribunarj.com.br/sinais-inteligentes-que-param-o-transito/. A Tribuna, 2018

[3] - https://oglobo.globo.com/rio/bairros/sinais-de-transito-inteligentes-comecam-operar-em-dezembro-em-niteroi-19676821. O Globo, 2016

[4] - https://oglobo.globo.com/rio/sistema-semaforico-de-22-anos-ainda-controlado-manualmente-no-rio-1-21805000. O Globo, 2017

[5] - https://eurio.com.br/noticia/10351/controle-do-trafego-em-duque-de-caxias-contara-com-semaforos-inteligentes.html. Eu Rio! , 2019

[6] - Natafgi, M. B., Osman, M., Haidar, A. S., & Hamandi, L. (2018). Smart Traffic Light System Using Machine Learning. *2018 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET)*. doi: 10.1109/imcet.2018.8603041 Disponível em <https://ieeexplore.ieee.org/document/8603041>

[7] - Balaji, P. G., German, X., & Srinivasan, D. (2010). Urban traffic signal control using reinforcement learning agents. *IET Intelligent Transport Systems*, *4*(3), 177. https://doi.org/10.1049/iet-its.2009.0096 Disponível em <https://ieeexplore.ieee.org/document/5558886>

[8] - Kwatirayo, S. & Almhana, J. & Liu, Z.. (2013). Adaptive traffic light control using VANET: A case study. 2013 9th International Wireless Communications and Mobile Computing Conference, IWCMC 2013. 752-757. 10.1109/

IWCMC.2013.6583651. Disponível em <]https://www.researchgate.net/publication/261378588_Adaptive_traffic_light_control_using_VANET_A_case_study>

[9] - Nafi, Nazmus & Khan, J.Y.. (2012). A VANET based Intelligent Road Traffic Signalling System. Australasian Telecommunication Networks and Applications Conference, ATNAC 2012. 1-6. 10.1109/ATNAC.2012.6398066. Disponível em < https://www.researchgate.net/publication/261079855_A_VANET_based_Intelligent_Road_Traffic_Signalling_System>

[10] - Płaczek, B. (2014). A self-organizing system for urban traffic control based on predictive interval microscopic model. *Engineering Applications of Artificial Intelligence, 34*, 75-84. doi:10.1016/j.engappai.2014.05.004 Disponível em < https://www.sciencedirect.com/science/article/pii/S095219761400102X?via%3Dihub>

[11] - https://ops.fhwa.dot.gov/publications/fhwahop06006/ chapter_6.htm FWHA, 2017

[12] - Bernas, Marcin, et al. “A Survey and Comparison of Low-Cost Sensing Technologies for Road Traffic Monitoring.” *Sensors*, vol. 18, no. 10, 26 Sept. 2018, p. 3243, 10.3390/s18103243. Disponível em <https://pdfs.semanticscholar.org/158a/d4a4a0e3bed2c512d67048b42cde78ed2307.pdf>

[13] Fernández, P.; Santana, J.M.; Ortega, S.; Trujillo, A.; Suárez, J.P.; Domínguez, C.; Santana, J.; Sánchez, A., 2016 SmartPort: A Platform for Sensor Data Monitoring in a Seaport Based on FIWARE. Sensors 2016, 16, 417. Disponível em < https://www.mdpi.com/1424-8220/16/3/417>

[14] - http://opendata.5t.torino.it/get_fdt. Open Data Torino, 2019

[15] - https://data.cityofchicago.org. Chicago Data Portal, 2019

[16] - https://redis.io/topics/benchmarks. Redis, 2019

Apêndice 1:

```
{
  "CurrentState": “green",
  "RedInterval": 23000,
  "YellowInterval": 3000,
  "GreenInterval": 46000,
  "LastStateUpdate": “14/05/2019-10:45:53.534234”,
  “PredictedNextStateUpdate”:
                    "14/05/2019-10:46:43.534234"
}
```

Apêndice 2:


```
{
  "IntersectionId": "90c4c8250987",
  "AreaId": "a135",
  "NeighboringIds": [
    "f99cdd2d",
    "e25c85ae",
    "059e45ed",
    "57c7aef5"
  ],
  "Configuration": {
    "South": {
      "In": null,
      "Out": {
        "Sensors": [
          {
            "Type": "speed",
            "id": "ca9c4bcc"
          },
          {
            "Type": "induction",
            "id": "06e410a2"
          },
          {
            "Type": "presence",
            "id": "b7db1540"
          }
        ],
        "TrafficLight": true,
        "Crosswalk": true,
        "Lanes": "2"
      }
    },
    "North": {
      "In": {
        "Sensors": [],
        "TrafficLight": false,
        "Crosswalk": false,
        "Lanes": "2"
      },
      "Out": null
    },
    "East": {
      "In": {
        "Sensors": [
          {
            "Type": "presence",
            "id": "2fe4ce51"
          }
        ],
        "TrafficLight": true,
        "Crosswalk": true,
        "Lanes": "2"
      },
      "Out": {
        "Sensors": [
          {
```

```
            "Type": "speed",
            "id": "b3b27c73"
          },
          {
            "Type": "induction",
            "id": "8ef48e99"
          },
          {
            "Type": "presence",
            "id": "2fe4ce51"
          }
        ],
        "TrafficLight": true,
        "Crosswalk": true,
        "Lanes": "3"
      }
    },
    "West": {
      "In": {
        "Sensors": [],
        "TrafficLight": false,
        "Crosswalk": false,
        "Lanes": "3"
      },
      "Out": {
        "Sensors": [
          {
            "Type": "speed",
            "id": "5c398210"
          }
        ],
        "TrafficLight": false,
        "Crosswalk": false,
        "Lanes": "2"
      }
    }
  }
}
```


Apêndice 3:


```
{
  "IntersectionId": "90c4c8250987",
  "CurrentView": {
    "South": {
      "In": {
        "CurrentState": "red",
        "PeopleEstimate": 0,
        "StoppedCars": 2,
        "CarsPassed": 8,
        "RedInterval": 23000,
         "YellowInterval": 3000,
         "GreenInterval": 46000,
        "LastChange": "14/05/2019-10:45:53"
      },
      "Out": null
    },
```

```
    "North": {
      "In": null,
      "Out": null
    },
    "East": {
      "In": {
        "CurrentState": "green",
        "PeopleEstimate": 3,
        "StoppedCars": 0,
        "CarsPassed": 12,
        "RedInterval": 32500,
         "YellowInterval": 3000,
         "GreenInterval": 25000,
        "LastChange": "14/05/2019-10:45:48"
      },
      "Out": {
        "CurrentState": "green",
        "PeopleEstimate": 3,
        "StoppedCars": 0,
        "CarsPassed": 23,
         "RedInterval": 34500,
         "YellowInterval": 4000,
         "GreenInterval": 12000,
        "LastChange": "14/05/2019-10:45:48"
      }
    },
    "West": {
      "In": null,
      "Out": {
        "CurrentState": null,
        "PeopleEstimate": 0,
        "StoppedCars": 0,
        "CarsPassed": 12,
        "RedInterval": 56500,
         "YellowInterval": 3000,
         "GreenInterval": 19000,
        "LastChange": "14/05/2019-10:45:48"
      }
    }
  },
  "TimeStamp": "14/05/2019-10:46:27"
}
```