Problema 2 - Névoa das Coisas

Daniel Fernandes Campos, João Pedro Rios Carvalho

dfc152@gmail.com, jprcarvalho1@gmail.com

Resumo. *Este relatório descreve o processo de desenvolvimento de um sistema de monitoramento de COVID-19 de alto desempenho, com funcionamento distribuído e escalável, provendo redução de atrasos e de dados na nuvem. Ao longo dele, foram necessárias as utilizações de conceitos sobre IoT, Fog Computing, MQTT e API RESTful para alcançarmos o objetivo proposto.*

1. Introdução

O sistema consiste em um protótipo de monitoramento de COVID-19, o qual conta com dispositivos que fazem medições constantes de indicadores do estado de saúde (Frequência Cardíaca, Frequência Respiratória, Pressão, Oxigenação e Temperatura Corporal) de um paciente. Este sistema é aberto a usuários do Sistema Único de Saúde (SUS), e por isso contará com milhões de usuários, e por isso precisa contar com uma ótima infraestrutura, capaz de lidar com a grande quantidade de usuários de maneira rápida e eficiente durante seu funcionamento. Visando isso, utilizamos a Computação em Névoa (Fog Computing) e o protocolo de comunicação Message Queuing Telemetry Transport(MQTT), o qual se baseia no *Machine-to-Machine* (M2M), além de uma Interface de Programação de Aplicações do tipo Transferência Representacional de Estado (API RESTful) para padronizar a comunicação na disponibilização dos dados dos pacientes. Tais conceitos serão melhor abordados na seção de Fundamentação Teórica deste relatório.

Por fim, o sistema também deverá conter uma interface gráfica para visualização dos dados das medições feitas para cada paciente, estando ordenados do paciente de maior gravidade para o de menor gravidade, com a quantidade de amostra sendo especificada pelo médico operador do sistema. Por se tratar de um protótipo, foi utilizado um sistema gerador de dados e pacientes para testar e validar o desempenho do sistema antes de sua inauguração.

2. Fundamentação Teórica

Esta seção se dedica a apresentar conceitos importantes e que foram essenciais para o desenvolvimento desse projeto, entre eles podemos destacar Internet das Coisas, Computação em Névoa, MQTT e API RESTful.

2.1. Internet das Coisas(Internet of Things - IoT)

A Internet das Coisas é um conceito de redes recente, que surge junto com os demais avanços tecnológicos da globalização mundial, e vem ganhando cada vez mais força com as automatizações dos mais diversos aparelhos eletrônicos integrados na rede, presentes em casas inteligentes, e até mesmo cidades inteligentes. Mas, a ideia em que se baseia não é tão recente, segundo Filho (2016), a internet das coisas tem suas raízes moldadas na *Radio Frequency Identification* - RFID, a qual ganhou fama na Segunda Guerra Mundial, sendo utilizada para identificar se aviões que capotaram eram aliados ou inimigos. Após a guerra, o conceito de RFID começa a ganhar força comercial na sociedade comum. Só em

2002 que surge o nome “Internet of Things”, em 2008 acontece a primeira conferência internacional sobre tal tema. Atualmente a IoT está cada vez mais presente na sociedade, com uma quantidade crescente de novos objetos eletrônicos se interligando também com o “big data”, já que, além de realizar as suas funções básicas, eles podem ser programados para a obtenção de dados, gerando muitos dados para a rede, os quais geralmente são transferidos principalmente para a nuvem, a fim de serem tratados para serem utilizados.

2.2. Computação em Névoa

Com o avanço crescente e popularização da IoT teremos cada vez mais dispositivos realizando essas funções e consequentemente, mais dados sendo gerados, os quais vão para nuvem serem tratados, mas, terá um ponto em que isso se tornará inviável devido à grande quantidade de dados e a velocidade com que alguns precisam ser tratados. Dessa forma, surge uma nova forma de pensar nessa logística, com a Computação em Névoa, ligada intimamente com a IoT.

“Computação em Névoa(Fog Computing) visa disponibilizar serviços da nuvem em localidades próximas dos clientes, de forma, que permite escalabilidade, redução de latência e otimização de recursos, efetuando processamento remoto e evitando que dados desnecessários sejam enviados para a nuvem.”(CASTILHO E KAMIENSKI). Assim, podemos perceber que a ideia central é descentralizar as funções de processamento de dados e tomada de decisões que ocorre na nuvem para dispositivos intermediários, que estão entre o usuário final e a nuvem, estando, preferencialmente, o mais perto possível do usuário final, para otimizar o tempo de resposta dos dispositivos e economizar dados na nuvem, e como resultado evitaria sobrecarga e congestionamento na mesma.

2.3. Machine-to-Machine(M2M)

Como apresentado por Aijaz (2014) *Machine-to-Machine* (M2M) é um paradigma de comunicação emergente que permite que dispositivos se comuniquem de forma autônoma, sem interferência de um humano. Ainda segundo o autor, este tipo de comunicação se difere de comunicações “convencionais” (se referindo a comunicação *Human-to-Human*) pelo fato do grande número de dispositivos conectados, pequenos pacotes de transmissão de dados, dentre outros.

Segundo Atzori, Iera e Morabito (2010) o IoT tem como visão uma rede global de dispositivos conectados com identidades e personalidades virtuais operando em espaços inteligentes e usando interfaces inteligentes para se comunicar em contextos sociais, ambientais e de usuário. Nesta visão, teríamos milhares de dispositivos conectados se comunicando de forma automatizada e inteligente, por exemplo, um usuário escolher um conteúdo para assistir e por onde ele se mover na casa o conteúdo seria transmitido automaticamente e de forma sincronizada para um aparelho de vídeo presente naquele quarto, fazendo ajustes nos aparelhos de som para que a experiência seja a melhor possível, ou, o monitoramento de milhares de cultivos de uma determinada planta ao redor do mundo de forma 100% autônoma.

2.4. *Message Queuing Telemetry Transport*(MQTT)

Message Queuing Telemetry Transport (MQTT) é um protocolo para transmitir dados entre componentes, através da rede, na forma de mensagens. Segundo

PAESSLERAG (2019) ele foi originalmente desenvolvido pela IBM em 1999 para ser um protocolo de comunicação leve e que não necessitasse de tantos recursos, porque era originalmente utilizado para monitorar tubulações de óleo através de uma conexão por satélite. Em 2014, o MQTT se tornou um padrão OASIS(Organização para o Avanço de Padrões de Informação Estruturada, do inglês, Organization for the Advancements of Structured Information Standards)

O padrão MQTT estabelece a comunicação entre dois dispositivos através de mensagens enviadas e recebidas, e pode ser melhor explicado dividindo-o em quatro tópicos, conforme PAESSLERAG (2019).

2.4.1. *Publishers* e *Subscribers*

Estes são os componentes que se comunicam entre si. Publishers são aqueles que enviam mensagens. Tomando como exemplo o protótipo que estamos desenvolvendo, o dispositivo de medição está constantemente enviando mensagens informando as medições realizadas, assim, ao realizar esse envio ele estaria sendo um *publisher*. *Subscribers* são aqueles que recebem mensagens, novamente com o exemplo do protótipo, os servidores estarão ouvindo as mensagens enviadas pelos dispositivos, assim, elas estariam agindo como *Subscriber*. Um componente pode ser ao mesmo tempo um Publisher e um Subscriber, estando ouvindo determinados tópicos e enviando mensagens em outros.

2.4.2. Tópicos

Tópicos são a forma de organizar e estruturar as mensagens que vão ser trocadas entre os dispositivos. Quando os *publishers* forem publicar uma mensagem, eles têm que especificar um determinado tópico no qual publicar e para receber essas mensagens, os *subscribers* têm que se inscrever nestes tópicos.

Os tópicos no protocolo MQTT são organizados por seções de conteúdo, e cada sessão é separada por um "/". Por exemplo, "main_server/new_fog/fog_da_bahia". Quando um cliente do MQTT vai se subscrever em um tópico pode ser usado *wildcard* para que o mesmo receba todas as mensagens que passem por um determinado pedaço de caminho, usando o caracter “#” após o “/”, como ao se inscrever em “main_server/new_fog/#” assim este cliente estaria se inscrevendo para receber todas as mensagens cujo tópico começam com “main_server/new_fog/” (ou seja ele estaria recebendo tanto “main_server/new_fog/fog_1”, quando “main_server/new_fog/fog_2”, quando todas que começam com a string “main_server/new_fog/”).

2.4.3. Broker

É a unidade mediadora de todas as mensagens enviadas através do MQTT. O Broker é um servidor que conecta os *publishers* e os *subscribers*, ele recebe todas as mensagens enviadas pelos *publishers*, e é o responsável por repassar estas mensagens para os devidos *subscribers*, então, todos os componentes devem estar conectados ao Broker, seja para enviar ou receber mensagens.

2.4.4. Qualidade do serviço(QoS)

Diz respeito à confirmação de envio e recebimento de mensagens dos *publishers* e *subscribers*. Para QoS 0 não é exigido confirmação, no QoS 1 o broker envia 1 mensagem indicando que recebeu a mensagem enviada, garantindo que a

mensagem chegou pelo menos uma vez, e no QoS 2 é enviada uma confirmação que a mensagem chegou e caso a confirmação não aconteça (ou seja, a mensagem seja perdida) o cliente continuará a publicar até receber esta confirmação, com QoS 2 a confirmação é feita através de um "*handshake"* para garantir que a mensagem foi recebida exatamente 1 vez, ou seja, para impedir que hajam mensagens duplicadas sendo recebidas.

2.5. Interface de Programação de Aplicações(API)

A Interface de Programação de Aplicações(API) é um serviço intermediário com padrões estabelecidos que permite ao cliente acessar recursos de serviços web através de suas definições e protocolos, permitindo uma melhor comunicação entre eles. A arquitetura mais usada é a REST ou Transferência de Estado Representacional, a qual define algumas restrições para a comunicação, como, arquitetura cliente/servidor com solicitações HTTP utilizando alguns formatos conhecidos no envio de dados, como o json, interface uniforme e bem definida, camadas de segurança e proteção de dados no servidor, entre outros.

Dessa forma, podemos definir uma API REST como uma interface que provê comunicação entre cliente e servidor através do HTTP, com uma formatação de dados, cabeçalhos e parâmetros bem definidos, na qual o cliente faça requisições e o servidor disponibilize tais informações solicitadas.

3. Desenvolvimento

Esta seção está dedicada a detalhar sobre o processo de desenvolvimento do sistema, explicando seu funcionamento, as decisões tomadas durante o período, e como ocorreu a integração entre suas diferentes partes.

3.1. Sistema Geral

Como objetivo geral do protótipo tem-se a capacidade de visualização dos N pacientes mais graves, com N sendo uma quantidade especificada através de uma interface gráfica, também é possível a fixação de um paciente escolhido. Outro ponto é a constante atualização do sistema, tanto dos dados dos pacientes, como também a ordenação dos N pacientes mais graves.

Para isso, dividimos o sistema em 4 subsistemas, são eles: Módulo Gerador, Servidor Intermediário(SI), Servidor Principal(SP) e Página Web, os quais funcionam na forma de micro-serviços integrados para fornecer maior escalabilidade, flexibilidade e manutenibilidade. A seguir, detalharemos o funcionamento de cada um deles e seus propósitos como módulos.

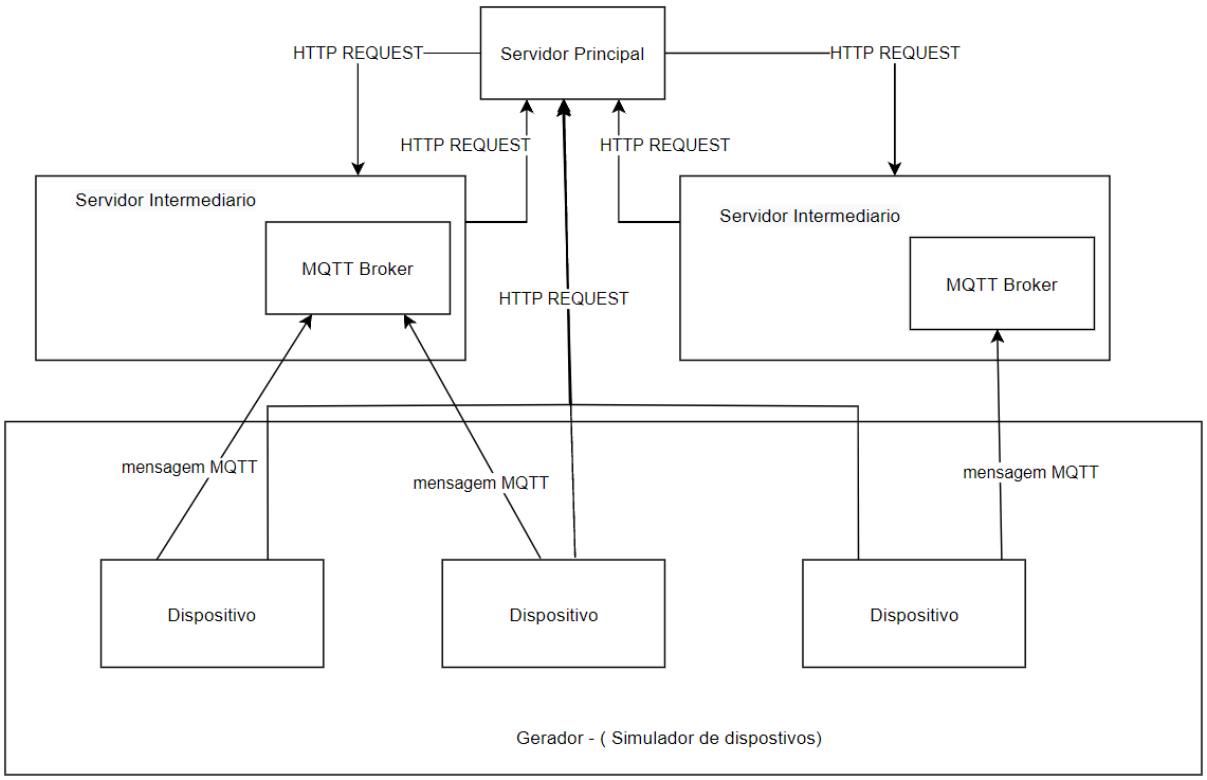

Figura 1: Diagrama da Arquitetura do Sistema. Fonte: Próprio autor.

Na figura 1 podemos ver o diagrama que demonstra a arquitetura do sistema, que será explicado, cada módulo, nos subtópicos seguintes e a forma como estes subsistemas interagem, e como se comunicam, no tópico seguinte.

3.1.1. Módulo Gerador

Este subsistema é o responsável por simular os dispositivos, nele os dispositivos são gerados de forma dinâmica, através da utilização de threads, assim, cada dispositivo tem seu funcionamento separado dos demais. Neste sistema, temos uma página web na qual o usuário pode estar criando dispositivos, indicando se a tendência deste dispositivo é transmitir os dados de um estado grave ou normal e cada dispositivo está fazendo envio de dados semi-aleatórios em um intervalo de tempo aleatório entre 0 a 2 segundos.

Para este funcionar temos que para cada dispositivo é dada uma tendência, como dito anteriormente, a qual define como os dados gerados aleatoriamente vão se comportar, podendo ser tendência ‘Grave’ ou ‘Normal’. Quando em tendência ‘Normal’, os dados ficam orbitando dentro dos parâmetros no qual o paciente é considerado normal, ou seja, nenhuma das medições indicam algum problema, já na ‘Grave’, as medições ficam orbitando dentro dos parâmetros de um paciente grave, ou seja, os dados das medições ficam alternando entre valores que indicam que o paciente tem algum problema.

Quando um dispositivo é iniciado independentemente da sua tendência, ele é gerado com dados aleatórios que podem, ou não, estar de acordo com a sua tendência, porém com o passar dos envios, este estado fica estável dentro da tendência enviada, existindo uma chance extremamente pequena do paciente ir para outra tendência de forma momentânea.

Na arquitetura desenvolvida para este problema também é responsabilidade do dispositivo fazer o envio da sua gravidade junto com os dados, assim, descentralizando parte do esforço computacional dos servidores intermediários, uma vez que o próprio dispositivo está fazendo o cálculo da gravidade, diminuindo o esforço computacional dos servidores intermediários, que não precisam fazer este cálculo para milhares de dispositivos. Esta Gravidade é dada pela fórmula arbitrária:

$$3 * (100 - \{Max\ Pressão\}) + 4 * (96 - \{Oxigenação\}) + \\ + 3 * (\{Frequência\ Respiratória\} - 20) + 4 * (\{Temperatura\} - 38) + \\ + 3 * (\{Frequência\ Cardíaca\} - 100) + 150$$

Então, seu resultado é arredondado para 2 casas decimais, sendo esta uma escolha arbitrária da equipe de desenvolvimento.

3.1.2. Servidor Intermediário

Para contemplar a ideia de *Fog Computing* foi criado um servidor intermediário com Python, entre o gerador e o servidor principal, por onde ocorre a passagem de dados no sistema. Quando inicializado ele comunica diretamente ao servidor principal e então é adicionado à lista de intermediários disponíveis, para que o gerador envie os dados dos dispositivos para seu respectivo servidor intermediário. Essa ação permite a criação e ativação de vários destes, e a qualquer momento da execução do sistema.

Ele opera recebendo os dados dos dispositivos do gerador, armazenando-os em um dicionário, ordenado automaticamente, e repassando, quando requisitado, ao servidor principal, os primeiros N mais graves, sendo N a quantidade de pacientes pedidos.

Também foi desenvolvida uma API REST em cada servidor intermediário, utilizando a biblioteca Flask, do Python. O objetivo foi de melhorar a performance na atualização do cartão presente na página web, uma vez que tal pedido é feito diretamente ao local que está armazenado tal paciente, sem que seja preciso passar pelo servidor principal.

3.1.3. Servidor Principal

No Servidor Principal temos de uma forma geral uma API REST, feita em Python e utilizando o Flask, nele é guardado uma lista dos Servidores intermediários cadastrados, guardando alguns dados destes servidores como IP e porta da API e IP e porta do broker MQTT, e a partir desta, os dados necessários são obtidos através de pedidos recursivos.

Neste Servidor são oferecidos a consulta dos N pacientes mais graves do sistema e consulta dos SIs que estão cadastradas no Servidor, retornando um Servidor Intermediário (usado pelos dispositivos ao serem iniciados, para mais explicações, ver subseção 3.2.2).

Ele atua constantemente pedindo os N pacientes mais graves de cada Servidor Intermediário cadastrado, e ordenando as listas recebidas para obter os N pacientes mais graves do sistema e armazená-los em uma outra lista. Então, na consulta dos N pacientes mais graves, ao receber este pedido, o servidor faz uma comparação do número de pacientes pedidos com a quantidade de pacientes requerida para cada SI, caso sejam iguais ou inferior, é retornado uma lista contendo os N pacientes pedidos, caso contrário, a quantidade de pacientes

pedidos é atualizada e é feito um novo pedido para os SIs com a quantidade atualizada, para, só então, retornarmos com a lista pedida. A ordenação é feita através da utilização de um SortedList da biblioteca SortedContainers presente no python, na qual é mantida uma lista ordenada que os itens são ordenados de forma crescente pelo oposto da gravidade, assim, os itens são adicionados de forma ordenada na lista e o seu tamanho permanece o maior requisitado para ficar pedindo aos SIs.

3.1.4. Página Web

Para a visualização do sistema temos um página web, desenvolvida com HTML e CSS, além de funcionalidades feitas com JavaScript, para automatização da mesma. Ela contém um campo de texto, uma tabela, indicando pacientes graves com uma marcação em vermelho, e um cartão de dados.

No campo de texto é possível especificar a quantidade de pacientes a serem observados, e então este será o número de pacientes contidos na tabela, mostrando o nome e a gravidade de cada um deles. Essa tabela se comunica a cada 4 segundos com o servidor principal, sendo atualizada e reordenada de acordo com os novos dados obtidos. Para fixar algum paciente específico, basta clicá-lo na tabela, e então, todos os seus dados(Frequência respiratória, Frequência Cardíaca, Pressão arterial, Oxigenação e Temperatura) serão mostrados no cartão, presente ao lado da tabela, junto com a identificação do paciente. Tais dados são atualizados a cada 2 segundos com pedidos ao servidor intermediário que eles estão. Mais detalhes sobre a dinâmica de comunicação do sistema como um todo serão mostrados futuramente neste relatório, através de diagramas e textos.

3.2. Comunicação do sistema

Nesta subseção falaremos sobre a comunicação das partes e como elas se integram, além das decisões tomadas.

Como dito anteriormente, o sistema é dividido entre Servidores Intermediários(SI), Servidor Principal(SP), Módulo Gerador e Página Web(Interface médica). Então, a comunicação entre SI e dispositivos gerados no módulo gerador se dá através do protocolo MQTT e todas as comunicações com o SP se dão através de requisições HTTP. Para explicar melhor essa comunicação foram preparados diagramas mostrando como se dão as comunicações.

3.2.1. Diagrama de comunicação MQTT

Como já explicado, cada SI terá o seu próprio broker MQTT e este é usado pelos dispositivos para realizar o envio de dados, assim, neste diagrama (Figura 2) temos a conexão dos dispositivos e o envio de uma mensagem, na rota de atualizar dados do paciente e como é o seu pacote.

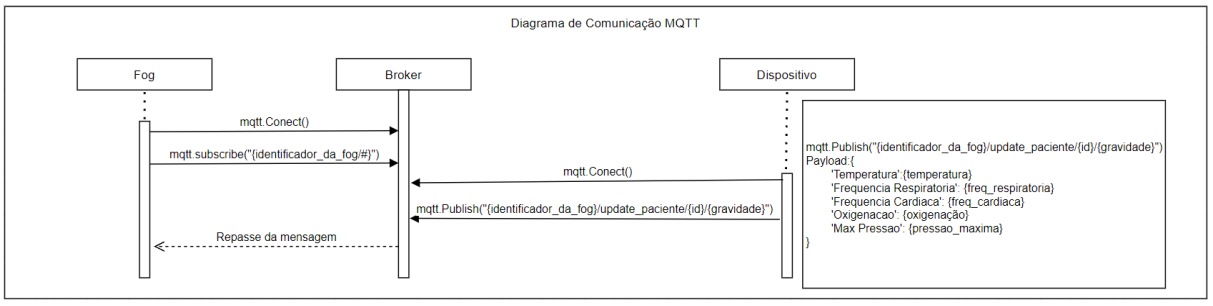

Figura 2: Diagrama de comunicação MQTT. Fonte: Próprio autor.

Para essa comunicação existe apenas um tópico usado, sendo ele: ‘{SI_id}/update_data/{identificador_do_paciente}/{gravidade}/{gravidade_anterior}’, o qual é usada para enviar os dados do dispositivo para os Servidores intermediários.

3.2.2. Diagrama de inicialização HTTP

Para fazer o controle de distribuição dos dispositivos com os SIs foi criado uma rotina de inicialização, na qual determina qual ação os SIs e dispositivos devem exercer ao serem inicializados. Desta forma, deixamos um embrião para utilização de filtros que direcionam dispositivos para SIs específicos seguindo uma lógica.

Na implementação realizada é passado um número, o qual é dividido pela quantidade de SIs cadastrados, então pegamos o resto desta divisão e direcionamos o envio do dispositivo para o SI com o índice do resultado. Para uma aplicação real poderia deixar esse filtro mais complexo, passando, por exemplo, a localidade e verificando qual SI está mais perto do dispositivo.

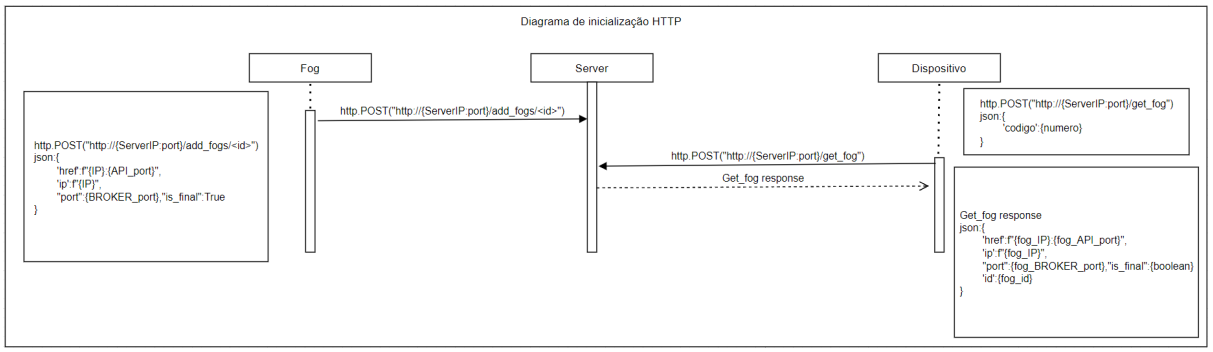

Figura 3: Diagrama de inicialização HTTP. Fonte: Próprio autor.

Neste diagrama vemos que ao iniciarmos um SI, ele manda uma requisição para o SP logo após ser iniciada, para se cadastrar na lista de SIs deste servidor e quando um dispositivo é iniciado, este manda uma requisição para o SP pedindo um SI, passando no pedido um json com os dados necessários para serem filtrados.

Quanto às rotas HTTP temos no Servidor Principal:

- ‘/pacientes/<quantidade>’: Rota para pegar N pacientes mais graves, sendo ‘quantidade’ um número pertencente ao conjunto dos Naturais.
- '/add_fogs/<id>': Para adicionar um SI ao SP, sendo ‘id’ uma string única que será usada para identificar o SI no sistema.
- '/get_fog': Para um dispositivo conseguir os dados necessários para se conectar a um SI.

Já nos Servidores Intermediários, temos:

- '/paciente/<id>': Para pegar os dados de um paciente específico, sendo ‘id’ o identificador do paciente. Neste sistema, o identificador corresponde ao nome do paciente (como o campo de identificação do paciente corresponde a uma string, este campo, ‘id’ da rota, também é uma string que deve ser única em cada SI, porém, sendo recomendado usar uma string única por sistema).
- '/pacientes/<quantidade>': Rota para pegar N pacientes mais graves, sendo ‘quantidade’ um número pertencente ao conjunto dos Naturais.

3.2.3. Diagrama de Comunicação Interface médica

A interface do médico mostra a lista de pacientes e também as medições sobre um dispositivo específico, para tal, são utilizadas as APIs disponibilizadas pelo SP e SIs, respectivamente, conforme o seguinte diagrama:

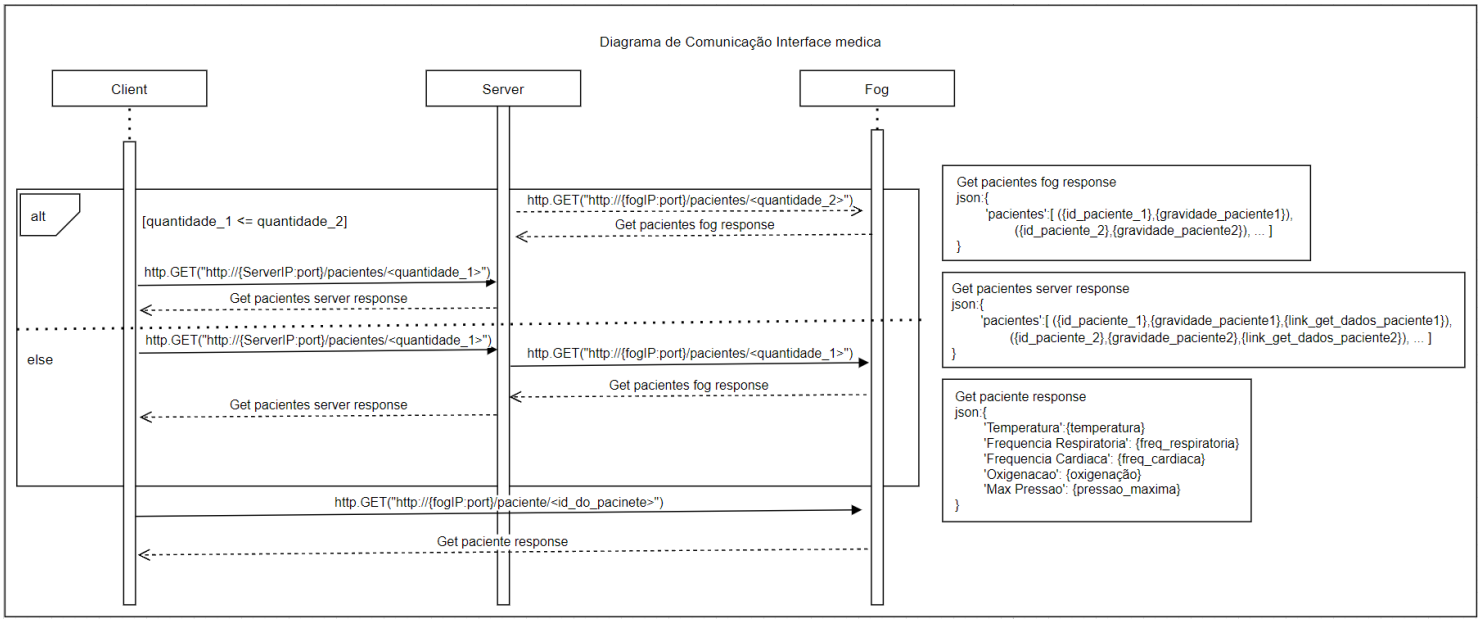

Figura 4: Diagrama Comunicação da Interface Médica. Fonte: Próprio autor.

Como pode ser visto no diagrama, para obter a lista de pacientes ordenada por gravidade, caso a quantidade pedida seja menor ou igual a quantidade de pacientes mantida na *cache* do servidor os dados são retornados desta *cache*, que é atualizada constantemente fazendo novos pedidos a cada meio segundo após o término do processamento anterior, caso essa quantia seja maior que a quantidade salva no sistema é feito uma consulta recursiva nos SIs cadastrados nele pedindo os N pacientes que lhe foi solicitado, assim, com os N pacientes de cada SI, eles são reordenados para ter-se os N pacientes mais graves do sistema como um todo e estes dados são salvos na *cache* e a requisição é respondida. Já para consultar as medições de um paciente, o pedido é feito diretamente para o SI, que retorna os dados deste paciente.

3.2.4. Diagrama de Comunicação Completo

Assim, juntando estes diagramas, temos um diagrama completo para a comunicação geral do sistema que mostra como todas as entidades do sistema comunicam entre si.

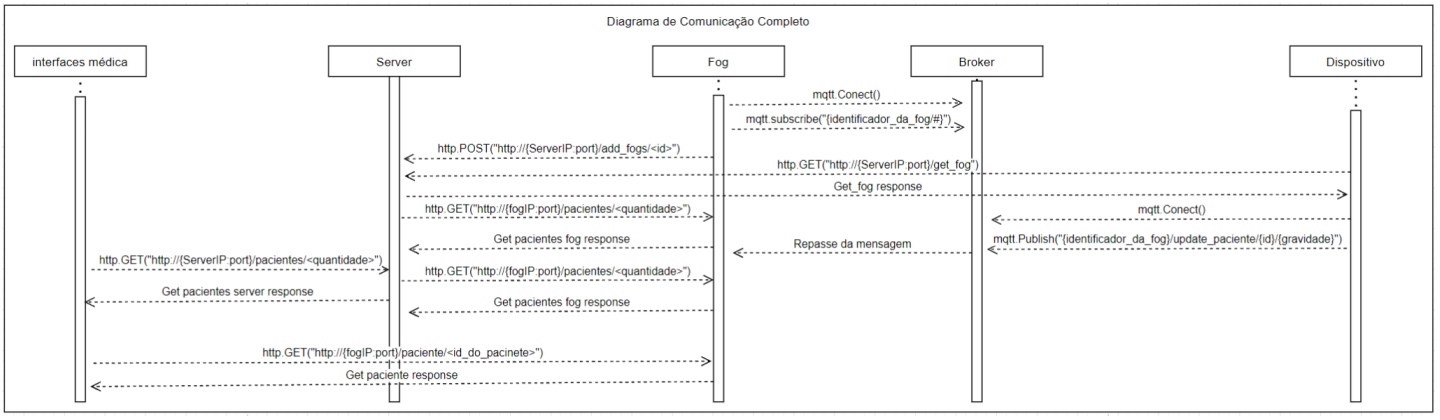

Figura 5: Diagrama de comunicação completo. Fonte: Próprio autor.

Neste diagrama vemos que a comunicação do SI com um dispositivo acontece através do protocolo MQTT e que cada SI tem o seu próprio *broker*, assim, para o dispositivo saber com qual SI ele deve se conectar, existe um protocolo de inicialização, o qual é feito através de requisições HTTP ao SP pedindo os dados do SI que ele deve se conectar (é necessário que o SI ao iniciar faça um envio de dados para o SP anunciando o seu funcionamento). Também podemos ver o fluxo de comunicação da interface, que foi explicado anteriormente.

3.3. Metodologia de testes

Nesta seção será explicado a forma, fórmulas e tecnologias utilizadas para medir o desempenho do sistema em relação a passagem de dados nas diferentes comunicações realizadas entre as partes do sistema.

3.3.1. Atualização de dados(do Dispositivo)

Na medição do delay de atualização das medições de um dispositivo foi calculado o tempo que demora para o sistema responder com os dados. Para tal, os testes são feitos adicionando um paciente ao sistema e fazendo requisições consecutivas até que a resposta contenha os dados atualizados. Para confirmação que os dados retornados são os atualizados e não dados enviados anteriormente, é enviado um campo adicional junto das medições indicando o número de sequência do envio, o qual ao recebermos a resposta da requisição é verificado se indica a medição mais atualizada. Assim feito os testes para o sistema com 1 a 1000 pacientes e são feitas 10 medições para cada quantidade.

Para tal, é feito o envio de medições através do MQTT para esta implementação e através de sockets na implementação anterior e então, é feito um requisição HTTP à API que retorna os dados de um paciente específico.

3.3.2. Performance da Página Web

Para medir o tempo de resposta, ou latência das requisições, foi utilizado o método ‘now()’ do objeto Performance do JavaScript, que é padrão na página web. Tal método retorna um *timestamp* relacionado à um instante de referência, ou seja, nos retorna um tempo, em milissegundos(ms), decorrido até o momento, tendo como base o instante de referência. Dessa forma, ele foi utilizado logo antes de requisitar dados, e logo depois da página receber a resposta, até mesmo dentro da “função de sucesso” da requisição(função que é configurada para executar logo após a requisição ter sido um sucesso).

Feitas as requisições e armazenados os *timestamps* falta então os cálculos necessários para medir a latência. Para isso pegamos o último *timestamp*(pego após a requisição) e subtraímos do primeiro(pego antes da requisição), e assim, tem-se o tempo de resposta de determinada requisição, em milissegundos. Este padrão é utilizado em cada requisição HTTP feita pela página web, seja para pegar os N pacientes mais graves no SP ou atualizar os dados de um determinado paciente presente em algum SI.

4. Resultados e discussões

Nesta seção mostraremos os resultados obtidos nos testes realizados e discussões sobre alguns pontos que consideramos importantes e/ou interessantes quanto aos mesmos.

4.1. Resultados testes para atualização de dados

Aqui falaremos sobre os resultados do delay do tempo de entrega, nestes testes foram enviados um pacote de dados junto com um campo indicando quando este pacote foi enviado, assim, ao chegar no sistema era feito o cálculo para determinar quanto tempo demorou para o pacote começar a ser processado. Assim, chegamos aos seguintes resultados:

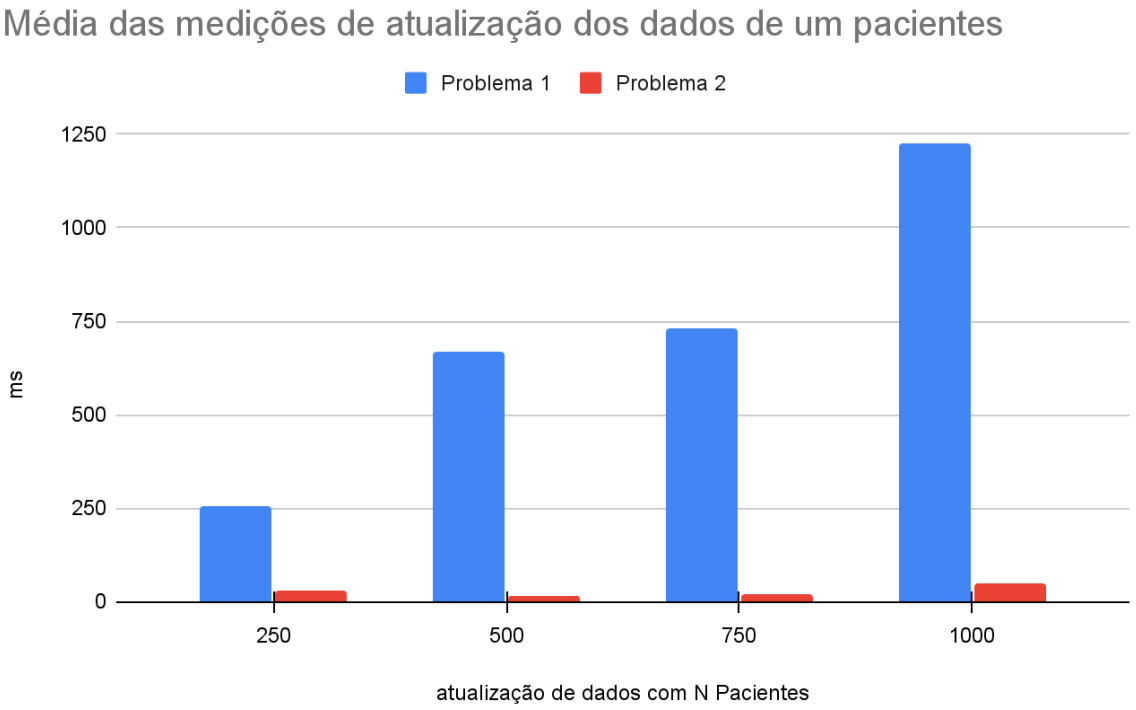

Figura 6: Médias dos testes de adição de pacientes para o problema 2. Fonte: Próprio autor.

Conforme o gráfico temos o número de pacientes que estão enviando os dados, no eixo X, e o tempo que demorou para os dados começarem a ser processados, no eixo Y. Assim, podemos ver quanto tempo demora para as requisições de atualização de dados acontecerem com N dispositivos enviando dados.

Assim, vemos que o tempo de envio de dados do dispositivo para o servidor no Problema 1 apresenta um aumento conforme existem mais pacientes no sistema e no Problema 2 esse aumento não foi observado.

4.2. Resultados testes de delay para display de dados de dispositivo

Neste foi medido qual o *delay* para termos os dados atualizados no sistema, como dito anteriormente, fazemos o envio dos dados e requisitamos os mesmos do servidor até que chegue uma resposta com os últimos dados enviados. Assim, os resultados podem ser vistos nas próximas duas figuras(Figura 7 e Figura 8):

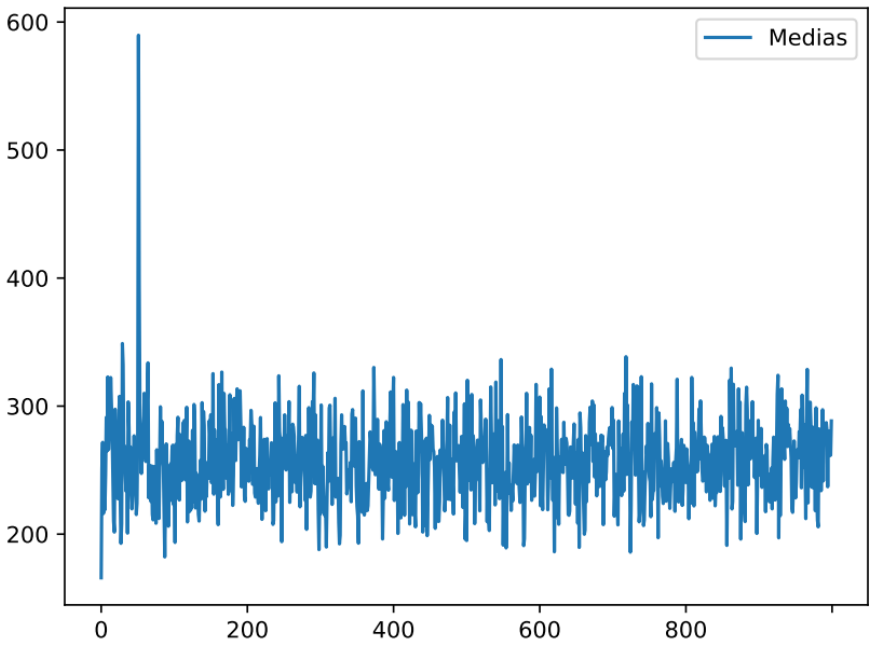

Figura 7: Médias dos testes de adição de pacientes para o problema 2. Fonte: Proprio autor.

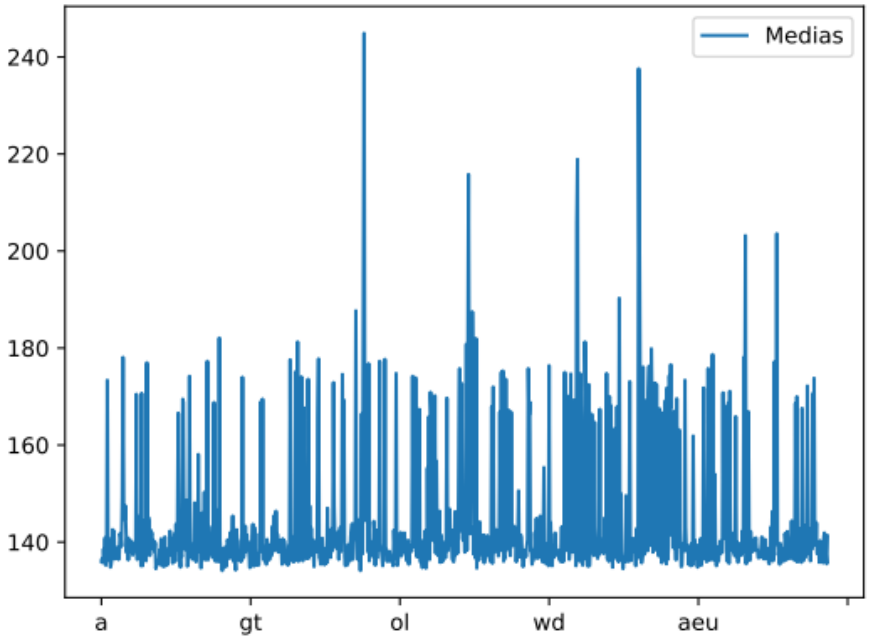

Figura 8: Médias dos testes de adição de pacientes para o problema 1. Fonte: Proprio autor.

Como pode ser observada nas figuras 7 e 8, temos no eixo X um identificador por paciente, na figura 7, um número, e na 8, uma string, e este, indica a quantidade de pacientes que já têm seus dados guardados no sistema. E no eixo Y temos o tempo que levou para a resposta com os dados mais atualizados acontecer. Assim, verificamos quanto tempo demora para o processamento total de uma única requisição com o sistema contendo dados de N pacientes salvos.

Conforme estes gráficos observamos que de uma forma geral o sistema não apresenta um aumento claro no tempo necessário para ter os dados mais atualizados. Assim, a média total de todas as medições para adição de pacientes no problema 2 foi de 256,80 ms com desvio padrão 31,80 ms, máximo sendo de 589,7 ms, mínimo de 168,9 ms. Já para as medições do problema 1 temos a média como

144,02 ms com desvio padrão 16,62 ms, máximo sendo de 244,9 ms, mínimo de 134,1 ms.

Assim, percebemos que o problema 1 tinha um tempo de atualização de paciente menor que o do problema 2. Consideramos que isso se dá pois no problema 1 os dados eram enviado do dispositivo direto para o servidor que iniciava uma *thread* para lidar com estes dados, já no problema 2, estes dados são enviados para o broker que então irá repassar para os Servidores intermediários, que então, irão colocá-los em uma fila para processamento.

4.3. Resultado dos testes de delay N pacientes mais graves

Para realizar as medições que serão abordadas nesta subseção, utilizamos a metodologia de testes apresentada na seção 3.3.2., e, a partir disso, pegamos a média dos valores e montamos o gráfico comparativo abaixo (Figura 8)

.

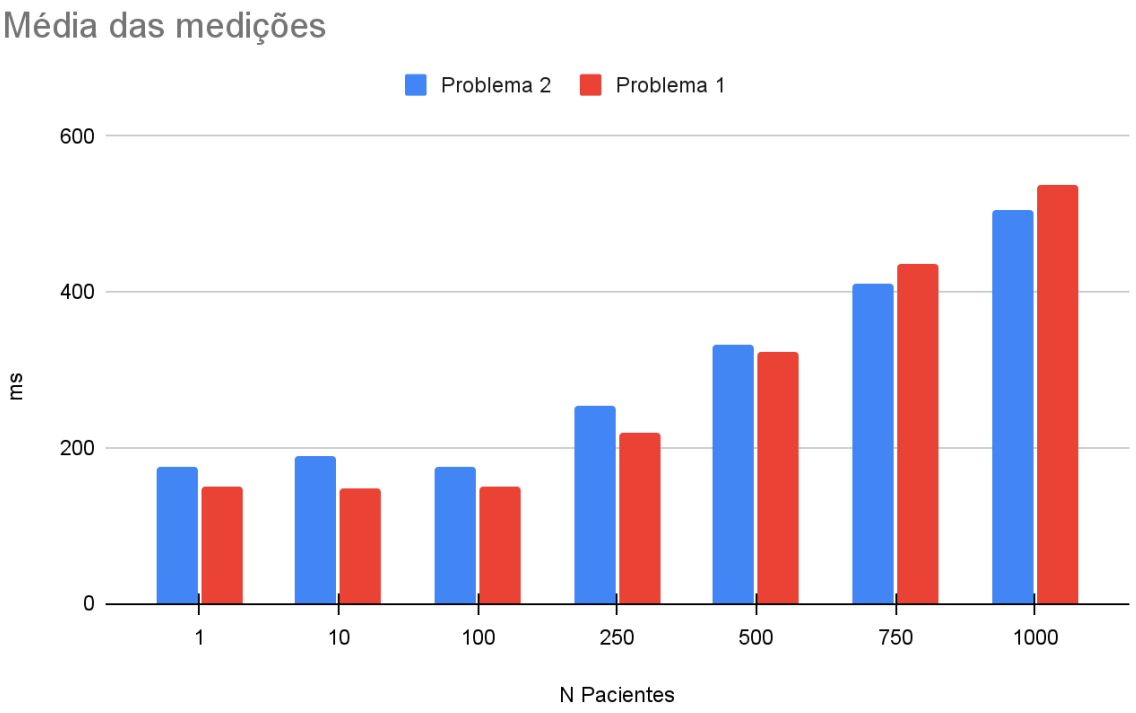

Figura 9: Gráfico das médias dos tempos de requisições. Fonte: Próprio autor.

No eixo X temos o número de pacientes requisitados da tabela, e no eixo Y o tempo de latência da requisição em ms. Analisando o gráfico percebemos que para um número pequeno de pacientes, até 250, o problema 1, especificado pela barra vermelha, apresenta melhor desempenho em relação ao problema 2, especificado em azul, mas que, a partir de 500 pacientes, os tempos começam a se igualar e mudar de paradigma, com o problema 2 passando a ter menor tempo de latência.

4.4. Discussões em geral

Ao vermos os subtópicos 4.1 e 4.2 pode gerar uma dúvida pois os dados parecem ser contraditórios, porém, no 4.1 temos o sistema inteiro em funcionamento, ou seja, os N pacientes do sistema estão enviando dados constantemente para serem

adicionados, já no 4.2, os dados são enviados somente do paciente que será testado a medição. Assim, no 4.1 testamos quanto tempo com o sistema em funcionamento demora para os dados serem recebidos e o processamento começar e no 4.2 testamos quanto tempo demora o processamento destes dados.

Já no 4.3. percebemos que nem sempre o problema 2 têm vantagem sobre o problema 1, isso ocorre por conta da mudança de arquitetura feita de um problema para o outro, onde adicionamos uma camada intermediária na comunicação, o que reduz um pouco o desempenho quando se trata de poucos pacientes, mas que, possibilitou uma maior escalabilidade quanto ao número de pacientes que o sistema suporta, e por isso, o problema 2 começa a apresentar melhor desempenho para um maior número de pacientes. É possível que, à medida que aumentássemos o número de pacientes, o problema 1 entraria rapidamente em colapso, enquanto que o problema 2 se manteria em funcionamento.

5. Conclusão

Com a realização deste projeto pudemos observar a importância e aplicação dos conceitos utilizados, nos dias modernos e no mundo globalizado, que necessitam, essencialmente, de sistemas com uma boa arquitetura, de fácil integração e de comunicação rápida e eficiente. Estes pontos são vitais, pois tornam possível comportar a grande massa de usuários e a transmissão de dados entre eles e/ou outros sistemas da melhor maneira possível. Dessa forma, foi possível atingir todos os objetivos propostos para o desenvolvimento, tanto em relação a implementação do problema, quanto a descoberta de novos paradigmas da programação.

Por esses fatores também foi possível concluir que a arquitetura implementada para solução do problema 2, como visto através das medições, é melhor que a solução do problema 1, pois percebemos que dada a escala de trabalho (pensando em milhões de dispositivos conectados) o problema 2 teria capacidade de suportá-los, uma vez que poderia ser adicionados novos SIs em tempo de execução, o que melhora ainda mais a escalabilidade da solução. Até mesmo, cada SI, individualmente, apresenta uma melhor performance em relação a outra implementação, quando está lidando com centenas de dispositivos atualizando seus dados constantemente.

Algumas possíveis melhorias seriam: implementação de filtros para poder ver os N pacientes mais graves de determinado Servidor intermediário (Na própria interface médica, uma vez que já existe a API que suporte tal operação), ou filtrando os SIs nos quais se deseja fazer esta busca; e a implementação de *multi-threading* na requisição dos N mais graves feita pelo SP para os SIs, o que adicionaria maior agilidade no tempo de atualização dos dados presentes no SP (uma vez que para milhares de dados não seria necessário esperar a resposta de um SI para pedir os dados para o próximo).

6. Referências

AIJAZ, Adnan. (2014). Protocol Design for Machine-to-Machine Networks. 10.13140/2.1.2786.7845.

ATZORI L. , Iera A., e Morabito G. 2010. “The Internet of Things: A Survey,” J. Comput.Networks, vol. 54, no. 15, pp. 2787 – 2805.

CASTILHO, Gustavo U., KAMIENSKI, Carlos A., “Aplicação de Computação em Névoa na Internet das Coisas para Cidades Inteligentes: da Teoria à Prática”. Disponível em: https://sol.sbc.org.br/index.php/wcga/article/view/2376/2340. Acesso em 13 out 2021.

FILHO, Mauro F. (2016), “Internet das coisas”, Disponível em: https://www.researchgate.net/profile/Mauro-Fazion-Filho/publication/319881659_Internet_das_Coisas_Internet_of_Things/links/59c038d5458515e9cfd54ff9/Internet-das-Coisas-Internet-of-Things.pdf. Acesso em 12 out 2021.

PAESSLERAG. 2019. What is MQTT? What you need to know. Disponível em: <https://www.youtube.com/watch?v=QSwR-JMmNO>. Acesso em: 18 out. 2021.

RED HAT, “API REST”. Disponível em: https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api. Acesso em 13 out 2021.