Pânico: sistema antipânico para hospitais públicos

Ivan Farina[1], Rodolfo Francisco de Oliveira[2]

[1]Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP) Campus Hortolândia – SP – Brasil

[2]Área de Informática – Campus Hortolândia – Instituto Federal de Educação, Ciência e Tecnologia de São Paulo

ivan.farina@aluno.ifsp.edu.br, rodolfo.oliveira@ifsp.edu.br

Abstract. *With the frequent cases of violence that occurs in public hospital's offices, this project was developed with the goal of assisting help requests from health professionals during the consultations. This system was developed with the TCP protocol for communication between network devices. Some of the main characteristics are low latency in the delivery of messages, the use of free technologies and/or Open Source, compatibility with any computer with Windows Operating System or based on Debian/Linux, the program can be activated away from the computer using a Bluetooth button, all this can be used in any public health unit.*

Resumo. *Com a frequente violência ocorrida em consultórios de hospitais públicos, o presente trabalho foi desenvolvido com o objetivo de auxiliar o pedido de socorro por parte de um profissional da saúde durante os atendimentos. Foi desenvolvido um sistema que utiliza o protocolo TCP para a comunicação entre dispositivos em rede. Suas principais características são baixa latência na entrega das mensagens, tecnologias gratuitas e/ou Open Source (código aberto), compatibilidade com qualquer computador com Sistema Operacional Windows ou baseados no Debian/Linux, botão Bluetooth para que o programa possa ser acionado longe do computador, tudo isso para que possa ser usado em qualquer unidade de saúde pública.*

1. Introdução

Embora a agressão de pacientes para funcionários em hospitais seja algo pouco noticiado, é bem comum de acontecer. O estudo (Galdino et al., 2017) revela que a violência no ambiente de trabalho realizada contra profissionais de saúde é considerada um problema de saúde pública no Brasil e no mundo e que por mais que as agressões verbais e psicológicas sejam os tipos mais comuns de violência ocupacional, evidencia-se que os abusos físicos têm crescido significativamente, podendo chegar à prevalência de 58% entre os trabalhadores dos setores de emergência.

Como forma de ajudar no combate a violência em hospitais públicos, o seguinte trabalho foi desenvolvido com o objetivo de auxiliar o pedido de socorro por parte de um profissional da saúde durante os atendimentos feitos em consultórios, geralmente realizados com as portas fechadas. A proposta do trabalho é desenvolver um sistema que envia um pedido de socorro através de uma rede TCP/IP aos líderes da equipe hospitalar, para que estes possam intervir no atendimento, no momento em que o funcionário se sentir lesado. O pedido pode ser enviado clicando no ícone situado na

Área de Trabalho do computador ou por meio de um botão Bluetooth guardado no bolso do profissional.

Espera-se que o sistema possa enviar as mensagens com uma baixa latência. Além disso, é esperado que o sistema possa ser instalado em qualquer computador com sistemas operacionais Windows ou baseados no Debian Linux (como por exemplo: Ubuntu, Linux Mint, Zorin OS, etc). O sistema também contará com uma interface gráfica para administração e configuração do mesmo por parte dos administradores e um banco de dados onde será cadastrado todas as ocorrências.

O restante do trabalho é organizado da seguinte maneira: Na Seção 2 é apresentado a revisão bibliográfica dos assuntos necessários para o compreendimento do trabalho. Na Seção 3 são apresentados os principais trabalhos correlatos. Na Seção 4 é apresentado a metodologia usada para o desenvolvimento do sistema. Na Seção 5 é apresentado a proposta e a especificação abstrata do projeto. Na Seção 6 é apresentado como foram desenvolvidos os protótipos do sistema. Na Seção 7 são apresentados os testes. Por fim, na Seção 8, é apresentado a conclusão e perspectivas de trabalhos futuros.

2. Revisão bibliográfica

Para a correta compreensão desse trabalho, a seguir será abordado dois temas essenciais: a violência em hospitais e sistemas baseados na arquitetura cliente-servidor.

2.1. Violência em Hospitais

A violência de pacientes para com a equipe hospitalar é bem comum de acontecer. É possível encontrar diversos trabalhos relacionados que abordam o assunto, como em (Silveira et al., 2014), que entrevista 14 profissionais de enfermagem de um hospital universitário no Brasil e analisa os casos de violência ocorridos. Os resultados obtidos, nesses casos, relatam que as violências ocorreram devido a falta de material nas unidades, alta demanda de pacientes e a insuficiência de recursos humanos. O silêncio e a busca de ajuda com outras pessoas foram as medidas para lidar com a violência.

Outro estudo similar é apresentado em (Galdino et al., 2017), que entrevista 16 enfermeiros em dois hospitais de média complexidade, localizados no norte do Pará. Nesse trabalho foi identificado que os atos agressivos foram orquestrados, em sua maioria, por pacientes e familiares. Suas motivações estavam relacionadas, principalmente, à maneira com que os envolvidos se comunicaram.

Também é possível encontrar estudos sobre a relação entre a violência sofrida pelos trabalhadores e sua associação com a doença burnout e transtornos psíquicos menores. Através de um estudo transversal, (Pai et al., 2015) analisa o caso de 269 profissionais da equipe de saúde em um hospital da região sul do Brasil. Para a coleta dos dados foram usados o *Survey Questionnaire Workplace Violence in the Health Sector, Maslach Inventory Burnout e o Self-Report Questionaire*.

2.2. Arquitetura Cliente-Servidor

A Arquitetura Cliente-Servidor representa dois *hosts* (hospedeiros) dos quais um, o Cliente, requisita um serviço/aplicação fornecido pelo outro, o Servidor. Caso a requisição seja aceita, o Servidor retorna a resposta ao Cliente (Kurose & Ross, 2013), como é possível observar na Figura 1.

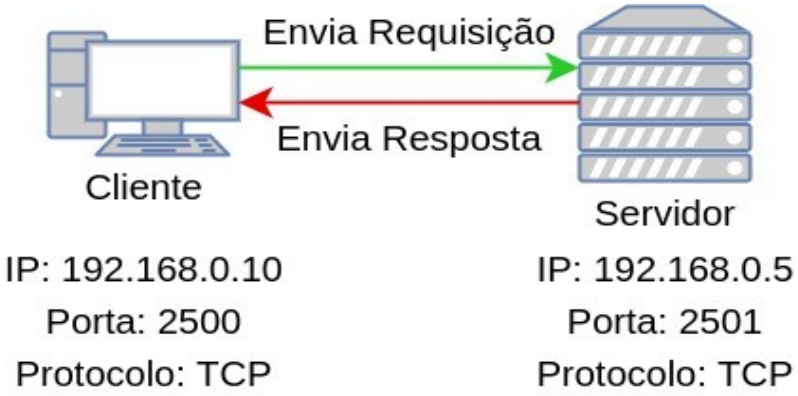

Figura 1: Arquitetura Cliente-Servidor

A comunicação entre os dois hospedeiros é realizada com base em um padrão chamado protocolo. Existem dois possíveis protocolos responsáveis por fazer essa comunicação: o protocolo TCP (*Trasmission Control Protocol* – Protocolo de Controle de Transmissão) ou o protocolo UDP (*User Datagram Protocol* – Protocolo de Datagrama do Usuário). O protocolo TCP é responsável por segmentar e reagrupar pacotes de dados, além de verificar a integridade dos mesmos, garantindo que todas as informações cheguem ao destinatário, sem perda de dados. Já o protocolo UDP não garante a entrega, nem verifica os dados. Dessa forma, sua vantagem é a velocidade na entrega do conteúdo, quando as mesmas chegam no destinatário (Neto & Raposo, 2014). O protocolo TCP está presente, por exemplo, em páginas web, e-mail, enquanto que o UDP está presente em jogos online, servidores DNS e protocolos de gerenciamento de rede.

Para que haja a comunicação Cliente-Servidor utilizando o protocolo TCP, é necessário que os dois hospedeiros se conectem. Já no protocolo UDP, o envio de dados ocorre sem a conexão (protocolo não orientado a conexão). A identificação das aplicações em rede, tanto cliente quanto servidor, é realizada através de um número de identificação conhecido como porta. A porta pode variar de 0 a 65.536, sendo TCP ou UDP (Sockets Linux, 2008). Por exemplo o serviço HTTP (*HyperText Transfer Protocol* - Procolo de transferência de HiperTexto) utiliza o TCP e é, por padrão, encontrado na porta 80. Já o serviço DNS (*Domain Name System* – Sistema de nome de domínio) utiliza o protocolo TCP para transferência de zona e o UDP para nome e consulta regular (primário) ou reverso na porta 53 (Windows, 2021).

Além disso, para que os dados sejam enviados através da rede, é necessário utilizar um protocolo chamado IP (*Internet Protocol*). O protocolo IP consiste em um datagrama contendo informações como a versão do protocolo e os endereços IP de origem e destino. As versões IPv4 e a IPv6 são as que estão ativas no momento e cada uma possui seu datagrama específico. Os endereços IPv4 são formados por 4 conjuntos de números de 0 até 255 denominados octetos, por exemplo: “192.168.0.1”. Dentro do endereço IP é possível saber a rede e o hospedeiro, com isso, a informação transmitida é repassada através dos roteadores até chegar na rede e hospedeiro especificados no datagrama (Kurose & Ross, 2013) conforme Figura 2.

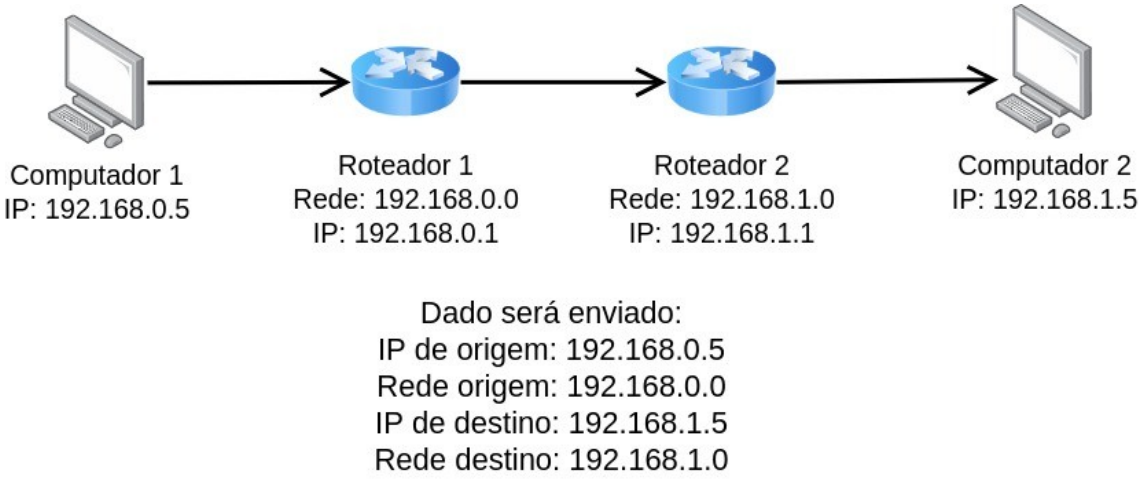

Figura 2: Exemplo de envio de dados de um computador a outro através da rede

O endereço IP e a porta representam o endereço completo, é como se o IP fosse o endereço de um prédio e a porta o número do apartamento. Porém, não há garantia que o serviço estará disponível no hospedeiro especificado. Para que a porta contenha um serviço é necessário um programa chamado *Socket*. O *Socket* funciona como uma API (*Application Programming Interface* – Interface de Programação de Aplicações) disponibilizada pelo Sistema Operacional para acessar as pilhas de protocolos (*Sockets Linux*, 2008). Ele é armazenado em uma estrutura de dados, no qual um dos campos dessa estrutura é o número da porta ao qual ele estará ligado. Dessa forma, para que um programa utilize o protocolo TCP é necessário implementar o *Socket* e dizer a qual porta ele será vinculado (O termo "vinculado" foi usado de uma tradução literal da palavra *bind* em inglês).

3. Trabalhos correlatos

Os trabalhos correlatos foram pesquisados nos periódicos CAPES, IEEE, no portal regional da BVS (Biblioteca Virtual de Saúde) e no JOEM (*Journal of Occupational and Environmental Medicine*) (termos utilizados na pesquisa: "*panic button in hospitals*", "botão pânico em hospitais", "protocolo de segurança contra violência em hospitais" e "violência em hospitais métodos de prevenção"). As pesquisas foram realizadas em agosto de 2021, porém não foi encontrado nada que abordasse um sistema antipânico para hospitais públicos. Devido a isso, foi buscado soluções comerciais que são relativamente relacionadas:

Primeiramente foi encontrada uma solução da empresa Bosh chamada "*Alarm over IP"* (Brinkman, 2006). Ela diz respeito a uma nova tecnologia, na qual, sistemas de segurança e alarmes de incêndio conseguem transmitir sinais de alarme, que antes eram baseados em telefone, através da comunicação TCP/IP. Isso fez com que diminuísse a latência na entrega da mensagem, aumento da disponibilidade do sistema devido a comunicação IP não interromper a linha telefônica, dentre outras vantagens.

Outra solução encontrada foi a *"EZ CALL IP – Nurse Call System"* (Communication, 2021). O objetivo dessa solução é chamar um enfermeiro quando o médico precisar de ajuda em algum atendimento ou o paciente pedir socorro quando internado. Ele funciona por meio de terminais que ficam com os enfermeiros e um *switch* principal que gerencia as requisições.

Também foi encontrada a solução *"Faint Detection Sensor"* (Security, 2021). Essa solução trata de um pedido de socorro para pessoas que estão prestes a desmaiar. Ela é usada principalmente em empresas que possuem funcionários portadores de deficiência. Essa ferramenta é acionada através de um sensor posicionado a alguns centímetros do chão, para que detecte pessoas que caíram no chão ou também pode ser usado através de um botão, interfone ou chamadas telefônicas. Depois de acionado, ele envia uma mensagem de socorro para a pessoa responsável por cuidar dessas situações na empresa.

Todas as soluções discutidas nessa seção são sistemas que se comunicam em rede através do protocolo TCP/IP, assim como o presente trabalho. Todas possuem seus diferencias com relação ao uso. A proposta apresentada no presente trabalho é a comunicação por meio de computadores que já estão instalados nos consultórios para o atendimento de pacientes, não sendo necessário comprar nenhum equipamento a mais. Além disso, foram utilizados apenas *softwares* gratuitos e/ou *open source* (código aberto) para seu desenvolvimento, justamente com o intuito de ser acessível a qualquer unidade de saúde pública. Outra vantagem, é que todas as ocorrências serão detalhadas e guardadas em um banco de dados, facilitando a pesquisa de profissionais que precisam saber sobre a violência nos hospitais.

4. Metodologia

A metodologia usada para o desenvolvimento do sistema foi a Prototipação Evolucionária com base na abordagem *Cornerstone* (Pedra Fundamental) como mostra a Figura 3. Essa metodologia sugere a criação e evolução de protótipos do sistema com o passar do tempo. Ela é muito usada para quando os requisitos do sistema não estão bem estruturados, sendo necessário realizar testes para estudar o projeto e evolui-lo com o tempo. A abordagem *Cornerstone*, diferente da *Throw-Away* (Jogar fora) evolui o protótipo estudado, consertando os defeitos e incluindo novos recursos pois ele fará parte do sistema final, enquanto que a *Throw-Away* requer o desenvolvimento de protótipos para somente estudo, sendo descartados logo após as devidas análises (Wazlawick, 2013).

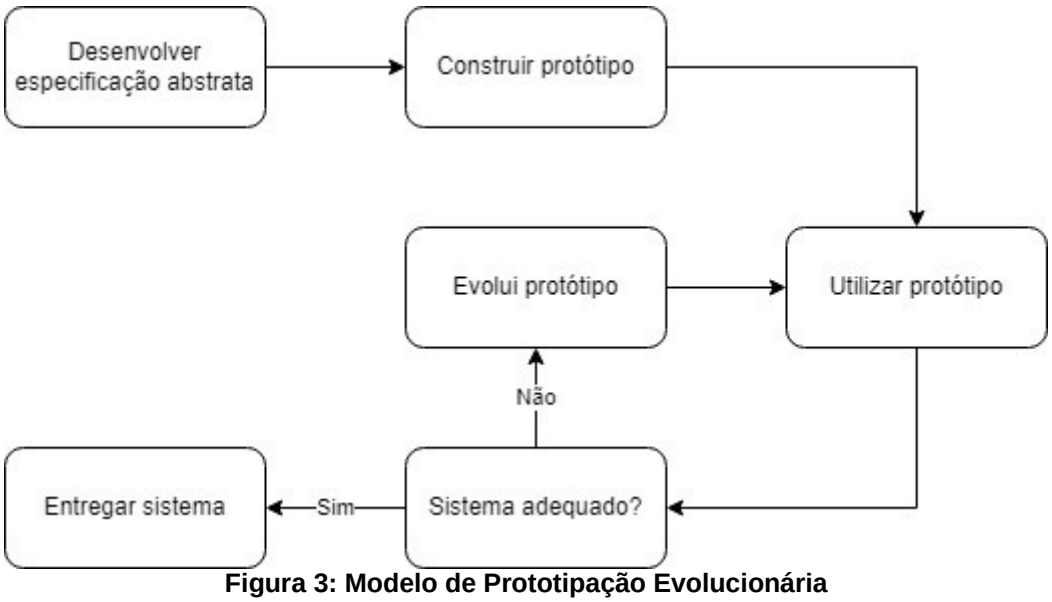

Figura 3: Modelo de Prototipação Evolucionária

5. Desenvolvimento do trabalho

Apresentada a metodologia, será apresentado, agora, o desenvolvimento do trabalho. Ele será descrito em tópicos especificando cada etapa do trabalho de acordo com a metodologia escolhida.

5.1. Desenvolver especificação abstrata – Apresentação da proposta

O presente trabalho tem como proposta solucionar um problema dentro dos hospitais com relação a segurança da equipe hospitalar no atendimento dos pacientes em consultas com as portas fechadas. O objetivo foi desenvolver um sistema que envia um pedido de socorro para os computadores da rede, no momento em que o funcionário se sentir inseguro ou em perigo. A Figura 4 apresenta os componentes do sistema, bem como a comunicação entre eles.

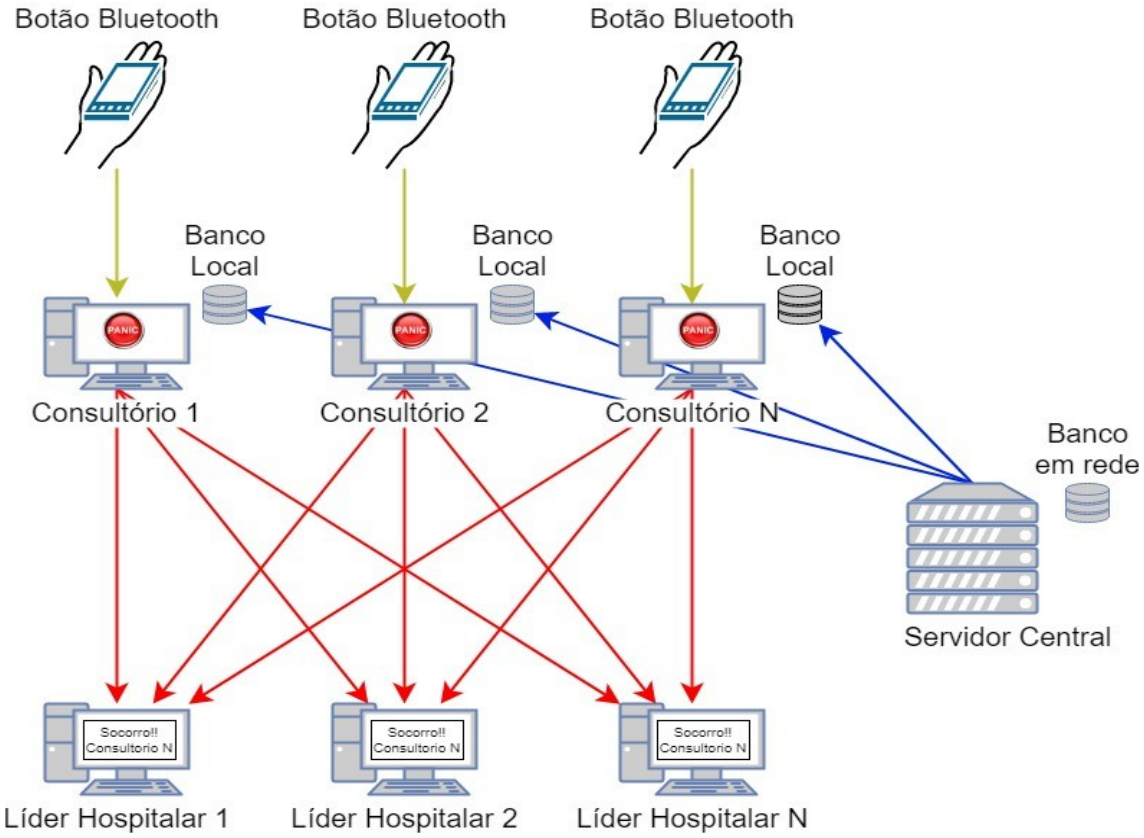

Figura 4: Diagrama de arquitetura da proposta

A seguir, será descrito cada um dos componentes apresentados na Figura 4:

- Consultório: esse é o componente que fica no computador do funcionário que pode estar em perigo. Ele foi configurado para enviar uma mensagem contendo a localidade do computador, por exemplo: “Consultório 1”, aos líderes com endereço IP cadastrados no Banco de Dados. A localização do consultório pode ser alterada, caso o computador seja transferido para outro local, através de uma *interface*

GUI (*Graphic User Interface*) local. A localidade do consultório é então salva, apenas, no banco de dados local. A mensagem de socorro pode ser encaminhada a partir de um botão externo ao computador (através de uma interface *Bluetooth*) ou por um ícone na Área de Trabalho.

- Líder Hospitalar: este módulo é instalado nos computadores que recebem o pedido de socorro. Ele abre uma conexão TCP para receber a mensagem através da rede. Uma característica importante é que tal módulo é inicializado junto com o sistema para que possa receber a requisição a qualquer momento enquanto o computador estiver ligado. A informação de socorro é exibida ao usuário através de uma GUI.
- Banco de Dados em Rede: esse banco armazena o endereço IP de todos os líderes hospitalares (ou seja, de todos os computadores que receberão o chamado de socorro), bem como todas as possíveis localidades dos consultórios (podendo ser chamado de outros nomes também, como por exemplo: sala). Além disso, conta com uma tabela que armazena os detalhes de todas as ocorrências onde foi necessário usar o botão. As tabelas contendo o endereço IP e as localidades são replicadas para os bancos de dados locais para fins de disponibilidade e baixa latência, todas as vezes que o Sistema Operacional dos computadores com módulo consultório é iniciado, através de um *script*. Os dados armazenados no Banco de Dados em Rede são mantidos através de uma interface CRUD (*Create Read Update Delete*), disponibilizada em uma GUI local, acessível do Servidor Central ou acesso remoto.
- Banco de Dados Local: para garantir baixa latência e disponibilidade, foi utilizado um banco de dados local na máquina em que o módulo consultório é instalado. Sendo assim, caso o banco de dados central fique inoperante, o módulo cliente poderá consultar o endereço IP dos servidores localmente. Ele será uma réplica do banco de dados central, possuindo uma tabela extra, a qual contém a localização atual (sala) do consultório.

5.2. Desenvolver especificação abstrata - Modelagem do sistema

A Figura 5 apresenta o diagrama de Casos de Uso do sistema. Nesse diagrama, é possível observar o Ator “Funcionário em perigo” no qual o mesmo terá a ação de pedir socorro através do sistema. Também é possível observar o Ator “Técnico em informática”. Este deverá manter (cadastrar, alterar e excluir) servidores, localizações, bem como alterar a localização atual do consultório. Além destes, o Ator “Líder hospitalar” que receberá o alerta de socorro e o Ator “Líder escritor da ocorrência” que além de receber o alerta de socorro será responsável por detalhar o ocorrido.

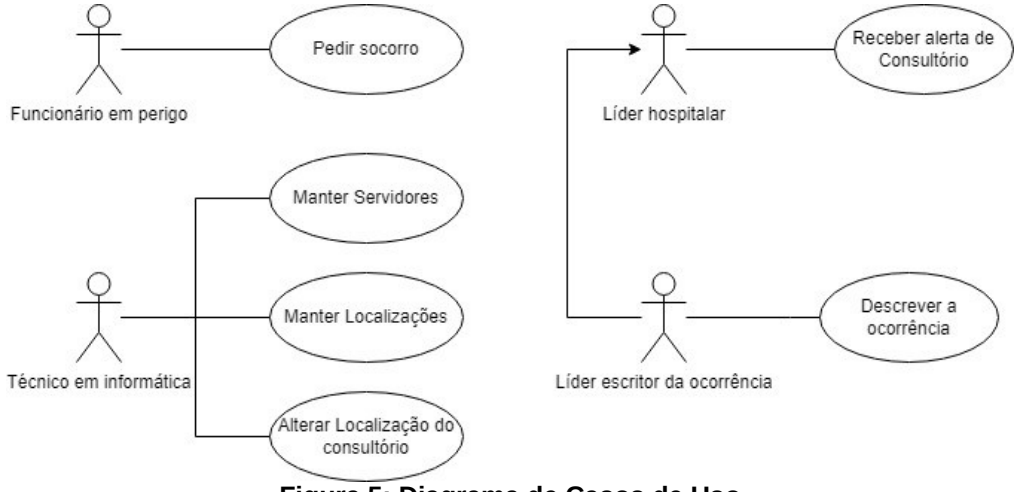

Figura 5: Diagrama de Casos de Uso

A Figura 6 representa o Diagrama Entidade Relacionamento (DER) do Banco de Dados em Rede. O banco de dados possui a tabela “Servidores” (seguindo o padrão da comunicação cliente-servidor), que armazena o endereço IP e a descrição do computador dos líderes hospitalares. Como não pode haver duas máquinas com o mesmo endereço IP na rede, foi usado o próprio IP como chave primária. A tabela “Todas_Localizacoes” armazena o nome de todos os possíveis ambientes onde os consultórios estão instalados. Como o nome dos consultórios são diferentes uns dos outros, foi utilizado o próprio nome como chave primária. Por último, a tabela “Ocorrencias”, que armazena a data de criação e os detalhes sempre que o botão é ativado. A chave primária ID é adicionada de forma automática. Os dados do atributo “detalhes” serão adicionados manualmente após o atendimento do funcionário em perigo. Seu intuito é conter os detalhes do ocorrido, para que, quando necessário pesquisar sobre a violência hospitalar, os dados de pesquisa estejam centralizados em um só lugar, não necessitando de entrevistas com funcionários violentados. Caso haja a necessidade de alterar os detalhes da ocorrência, será alterado automaticamente o valor do atributo “Data_alteracao” para a data atual da alteração.

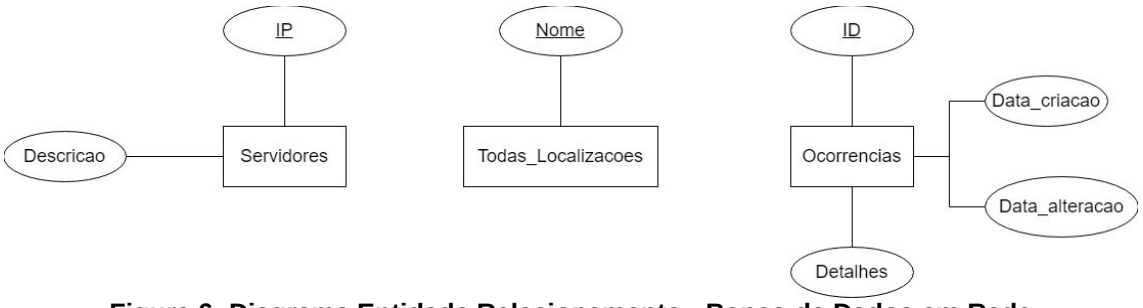

Figura 6: Diagrama Entidade Relacionamento - Banco de Dados em Rede

A Figura 7 representa o Diagrama Entidade Relacionamento (DER) do Banco de Dados Local. As tabelas “Servidores” e “Todas_Localizacoes” são replicações do Banco de Dados em Rede, dessa forma, elas possuem as mesmas especificações. Já a tabela “Localizacao_Atual”, armazena o nome da localização atual do computador. Ela é baseada na tabela “Todas_Localizacoes”. Como pode haver mais de um computador no

consultório, foi decidido não vincular as duas tabelas para ser permitido cadastrar mais clientes com a mesma localização.

Figura 7: Diagrama Entidade Relacionamento - Banco de Dados Local

O diagrama de classes foi desenvolvido contendo as classes DAOLocalizacaoAtual, DAOLocalizacoes, DAOOcorrencias, DAOReportarSocorro e DAOServidor conectadas a classe CriarBanco, no qual essa possui as funções de criação, alteração e exclusão de dados do banco. As classes InterfaceAlteraLocalizacaoAtual, InterfaceAlteraLocalizacoes, InterfaceAlterarOcorrencia, InterfaceAlterarServidor, InterfaceCadastroLocalizacoes, InterfaceCadastroServidor, InterfaceExcluiLocalizacoes, InterfaceExcluirOcorrencia, InterfaceExcluirServidor, InterfacePedidoSocorroComReport, InterfaceVisualizarLocalizacoes, InterfaceVisualizarOcorrencia, InterfaceVisualizarServidores, scriptAttBanco e a tcpCliente foram conectadas as classes DAO para o acesso aos dados e manipulação dos dados do banco. A classe InterfacePaginaInicial foi desenvolvida para chamar as interfaces de CRUD dos servidores e das localizações e a classe InterfacePaginaInicialReporte chama as interfaces de CRUD das ocorrências. A classe botao_bluetooth está conectada com a classe tcpCliente, para quando for acionada, executar o pedido de socorro. Para o recebimento das mensagens existem duas classes, a classe tcpServidor que está conectada com a classe InterfacePedidoSocorro, na qual apenas exibirá na tela o pedido de socorro, dessa forma não houve a necessidade de conectá-la a mais nenhuma outra e a classe tcpServidorComOcorrencia que está conectada a classe InterfacePedidoSocorroComReport, essa fara o cadastro dos detalhes da ocorrência, então como dito acima, está conectada a classe DAOOcorrencias para fazer o CRUD desses dados. Devido ao tamanho do diagrama de classes, para não comprometer seu entendimento, foi decidido disponibilizá-lo através de um link do GitHub, dentro desse link, além do diagrama de classes há também os códigos do sistema e uma breve explicação de como executá-los [Farina, 2021].

6. Construir sistemas protótipos

O desenvolvimento do sistema foi estruturado em cima das seguintes características: funcionar em qualquer sistema operacional, tanto Windows quanto baseados em GNU/Linux, usar tecnologias gratuitas, além disso, era necessário que o sistema não possuísse baixa latência e entrega de pelo menos 90% das mensagens. Outro ponto é que como o intuito era de implementar o sistema em hospitais públicos, o objetivo é que fosse leve e que não utilizassem programas externos para funcionar para que não fosse exigido possuir um hardware robusto para sua execução.

Devido a metodologia escolhida, Prototipação Evolucionária *Cornerstone*, foi possível desenvolver o projeto em *Sprints* e ir melhorando o sistema conforme fosse visto a necessidade, pensando sempre em manter as características descritas acima. A seguir será descrita cada *Sprint* e as evoluções que o sistema foi adquirindo.

6.1. *Sprint* 1

Na primeira *Sprint* do projeto foram levantados os requisitos do sistema e desenvolvido os diagramas DER, casos de uso e de classes para estruturar o que deveria ser desenvolvido e quais as prioridades.

Depois da ideia estruturada, a linguagem escolhida para codificar o sistema foi Python versão 3.8 (Python, 2021b), por possuir ampla documentação na Internet e dispor da classe Socket para manipular os sockets de conexão. A codificação deu início através do desenvolvimento das classes TcpCliente e TcpServidor, nos quais, o TcpCliente é o *script* que vai enviar o pedido de socorro e o TcpServidor é o programa que vai receber esse pedido. Dentro da classe *Socket*, foi usado o tipo SocketStream para o desenvolvimento de *Sockets* do tipo TCP.

6.2. *Sprint* 2

Com a conclusão dos *scripts* de envio e recebimento de mensagens, começou-se a planejar o armazenamento dos endereços IPs dos computadores que receberiam o chamado de socorro. Para isso, foram analisadas as tecnologias que seriam usadas para o desenvolvimento do banco de dados do sistema. Por haver a necessidade de baixa latência na entrega das mensagens, optou-se pelo desenvolvimento de um banco de dados local, para garantir baixa latência na comunicação. Dessa forma, a tecnologia escolhida foi a SQLite, pois, por gravar os dados diretamente em arquivos de disco, não é necessário instalar nenhum programa adicional, diminuindo a necessidade de um hardware robusto. Além disso, o mesmo é multiplataforma e possui baixa latência por buscar os dados diretamente no disco, sendo o banco de dados relacional mais rápido, se comparado com as outras opções (SQLite, 2021).

Com o banco de dados local definido, surgiu a necessidade de como criar o banco e como manipulá-lo. Para isso, foi usado a ORM (Mapeamento objeto-relacional) SQLAlchemy (SQLAlchemy, 2021). Com ela, foi possível construir as tabelas do banco usando o paradigma orientado a objetos com o Python.

As tabelas que foram geradas nessa *Sprint* foram a tabela Servidores que contém a descrição e o IP do computador que receberá o pedido de socorro e a tabela Localização_Atual, pois um dos requisitos funcionais era a possibilidade de alterar a localização do computador caso necessário. Esta tabela continha apenas o campo nome. Além disso, na instanciação da tabela era criado um valor padrão para, possivelmente, ser alterado com o passar do tempo, não sendo possível criar valores ou excluir o existente.

6.3. *Sprint* 3

Com o banco criado, começou-se a criação das GUI's do sistema para fazer a criação, alteração, exclusão e seleção (CRUD) dos dados do banco. Primeiramente, foi feito um estudo sobre as classes de interface gráfica que o Python possui. A primeira estudada foi a GTK. Com ela, é possível criar interfaces para diversas linguagens, pois trabalha em XML e para cada linguagem só é necessário sua importação. Para criar as GUI pode-se usar um software chamado Glade (GTK, 2021). Foi desenvolvida algumas interfaces,

como cadastro de servidores, alterar localização atual e alterar servidores. No entanto, quando foi realizado testes, descobriu-se que a GTK funciona nativamente nos sistemas GNU/Linux, mas em ambiente Windows é necessário a instalação do software MSYS2. Como o objetivo inicial era de que o sistema funcionasse em qualquer ambiente sem a necessidade de programas externos instalados, decidiu-se não levar esse recurso adiante.

A próxima da lista foi a Kivy. Ela é amplamente usada para o desenvolvimento de games através da linguagem Python. Ela possui diversas funcionalidades para manipulação da interface, porém é necessário instalar o software OpenGL. Essa API trabalha com ambientes gráficos que requerem um harwdare mais robusto, como animações 3D e simuladores visuais (OpenGL, 2021). Por esse motivo, este recurso também foi descartado.

A terceira e última foi a Tkinter. Ela funciona nativamente em ambientes Linux e Windows, de fácil manipulação e com vasta documentação disponível na Internet. O primeiro teste realizado com essa linguagem foi o desenvolvimento de uma interface gráfica para o *script* TcpServidor, como mostrado na Figura 8. Nesta interface o intuito era de no momento de recebimento do pedido de socorro a interface fosse para o topo da tela do usuário e mostrasse de forma objetiva o pedido de socorro com a localização do computador. A implementação foi um sucesso, cumprindo todos os requisitos relacionados a interface gráfica. Decidiu-se seguir com este recurso para o desenvolvimento das outras telas.

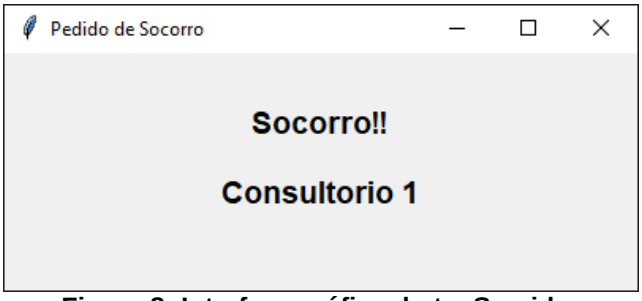

Figura 8: Interface gráfica do tcpServidor

6.4. *Sprint* 4

Depois da definição de qual classe seria utilizada para o desenvolvimento das interfaces, foram criadas as classes DAO (*Data Access Object*), que atribui e encapsula e abstrai os mecanismos de acessos ao banco de dados (AECE, 2005). Ela foi desenvolvida utilizando a classe SQLAlchemy. Nessa *Sprint* foram criadas as classes DAOServidor, DAOLocalizacaoAtual.

6.5. *Sprint* 5

Nesta *sprint*, foi realizado o desenvolvimento das GUI para cadastro, alteração, exclusão e visualização dos Servidores e a de alteração da Localização Atual. Um exemplo das interfaces criadas é a Figura 9 que mostra o resultado do cadastro de servidores. Após isso, cada interface foi vinculada com a classe DAO e utilizando os métodos dessa classe elas estavam prontas para o uso.

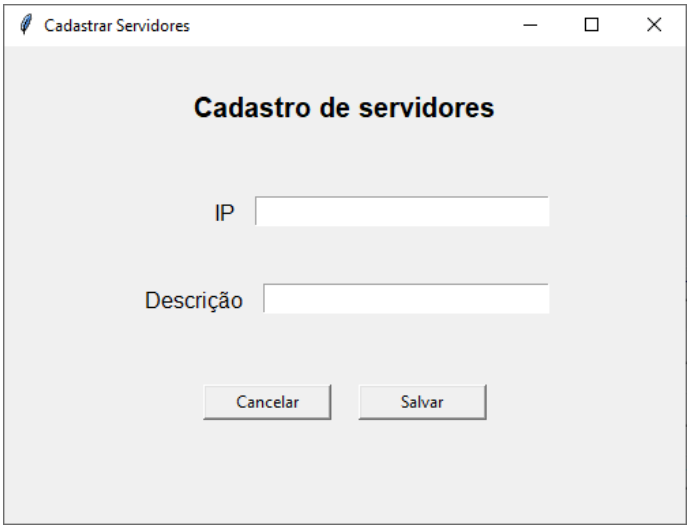

Figura 9: Tela de cadastro de servidores

6.6. *Sprint* 6

Nessa *Sprint*, surgiu a necessidade da criação de uma tabela chamada "Todas_localizacoes". Esta tabela seria encarregada de armazenar todos os nomes de localizações que o hospital receberá, para que a alteração da Localização Atual seja padronizada. Ela continha apenas o atributo nome da localização. Além da criação dessa tabela, foi desenvolvido também a classe DAO e as GUI de CRUD. Depois disso, foi criado uma interface central para chamar as interfaces de Servidor e de Localização, como representado na Figura 10.

Figura 10: Tela de gerenciamento de CRUD de servidores e Localizações

6.7. *Sprint* 7

Neste ponto, a ideia original do projeto estava praticamente concluída, mas após conferir o trabalho (Pai et al., 2015) onde os autores analisaram o caso de 269

profissionais da equipe de saúde através de um estudo transversal, decidiu-se acrescentar no escopo do projeto um meio de salvar as ocorrências, para facilitar possíveis estudos futuros sobre o tema de violência em hospitais. Dessa forma, foi criado uma nova tabela com o nome Ocorrencias que contém principalmente os atributos detalhes, data_criacao e data_alteracao, foi criada uma classe DAO para manipulação do banco de dados e as GUI foram disponibilizadas da seguinte forma: para o cadastro foi criado uma outra interface para o TcpServidor, como representado na Figura 11. Nessa nova interface há a possibilidade de descrever o ocorrido. Apenas um usuário terá acesso a essa interface, os demais deverão utilizar a interface comum do TcpServidor. Além disso, foi criado as interfaces de alteração, exclusão e visualização dos ocorridos.

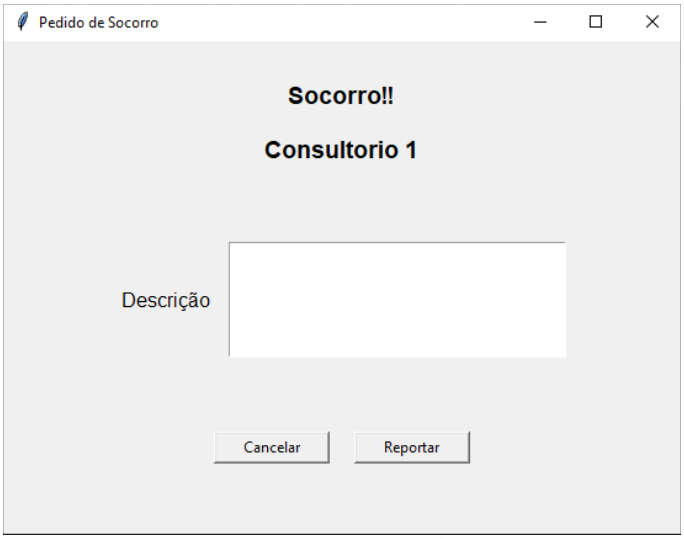

Figura 11: Interface gráfica do tcpServidorComOcorrencia para quem for escrever as informações da orcorrência

6.8. *Sprint* 8

Depois das configurações locais finalizadas, foi desenvolvido o *script* de atualização da base de dados local de acordo com a base de dados em rede. O *script* é executado toda vez que o computador reiniciar e seu trabalho é buscar os dados do banco de dados em rede. Caso haja algum valor nesse banco, ele vai apagar a tabela no banco de dados local e cadastrar os dados trazidos do banco de dados em rede.

6.9. *Sprint* 9

Nessa *sprint* foi desenvolvido o *script* relacionado ao botão Bluetooth. O dispositivo escolhido foi um Disparador Bluetooth presente em um “Bastão de Selfie”, representado na Figura 12. Inicialmente foi pensado em uma programação focada na conectividade Bluetooth. Dessa forma, foi criado um *script* parecido com o tcpServidor, utilizando para isso a classe “PyBluez”, para receber o sinal do dispositivo. Porém, o *script* não funcionou, pois é necessário outro *script* sendo executado do cliente para enviar os dados ao servidor. No entanto, o disparador Bluetooth usado não pode ser modificado

para enviar os dados necessários. Foi pesquisado, também, soluções usando sockets, mas por sempre necessitar de um cliente para a comunicação, esses tipos de sistema não funcionam. Após alguns estudos e análises, foi encontrado uma funcionalidade específica desse disparador que foi possível utilizar: a função de aumentar o volume. Quando o mesmo está pareado com um computador ele é reconhecido como um teclado. Então usando a classe “pynput” do Python, foi codificado o *script* que ao clicar na tecla do volume mais de duas vezes é executado o programa tcpCliente que enviará o pedido de socorro. Dessa forma, quaisquer disparadores Bluetooth que possuem essa funcionalidade de aumentar o volume do dispositivo poderão ser utilizados no sistema.

A Tabela 1 sintetiza as ferramentas utilizadas para o desenvolvimento do presente proposto.

Figura 12: Bastão de selfie

Tabela 1 – Ferramentas utilizadas para o desenvolvimento da proposta

Funcionalidade	Usado no projeto
Linguagem para codificação	Python 3.8
Classe para envio e recebimento das mensagens	Socket
Classe para criação de GUI	Tkinter
Classe para manipulação do banco de dados	SQLAlchemy
Banco de dados Local	SQLite
Banco de dados em Rede	Mysql

7. Testes

Para testar o sistema, foram realizados testes de disponibilidade, para saber qual a porcentagem de entrega do pedido de socorro que o software possui. Para a realização dos testes foram utilizados um computador com sistema operacional Windows 10, um *laptop* com sistema operacional Linux Mint 20.1, um disparador Bluetooth e dois roteadores de uso domiciliar que possuem *Switch* e *Access Point*. A rede LAN foi configurada com o *desktop* conectado via cabo Ethernet no roteador mais próximo, um roteador ligado ao outro via cabo Ethernet e o notebook conectado via Wi-Fi em um roteador de acordo com cada teste.

O primeiro teste foi realizado com o *desktop* localizado na sala de estar, o notebook no dormitório e o roteador mais ou menos centralizado entre os dois. A Figura 13 mostra uma planta do ambiente que foram concebidos os testes. Para a realização dos testes foi acrescentado um *loop* no código do *script* tcpCliente para enviar cem vezes seguidas o pedido de socorro. Também foi acrescentado no *script* tcpServidor um contador para saber quantas foram recebidos os pedidos. O primeiro teste foi realizado como o notebook sendo o tcpCliente e o computador sendo o tcpServidor. Além disso, o primeiro teste foi realizado utilizando a interface gráfica do tcpServidor. Toda vez que chegava um pedido de socorro, era aberto automaticamente a tela apresentada Figura 7. Em relação a interface gráfica foram observados dois pontos: primeiro: somente uma tela é aberta por vez, independente da quantidade de requisições que foram enviadas; segundo: o *script* armazena na memória uma segunda requisição que foi recebida na sequência, para que quando a tela da primeira for fechada a segunda abrir. Como ele só armazena uma próxima requisição na fila e só a abre quando a primeira for fechada, o computador “trava” o recebimento de requisições, então é necessário o monitoramento de um usuário para fechar as telas que forem abertas para que o fluxo de recebimento seja contínuo. No fim dos testes as cem requisições enviadas foram recebidas. A fim de averiguar o motivo do travamento a cada duas requisições recebidas, foi realizado o mesmo teste sem a interface gráfica. Nesse teste não houve travamento na entrega das requisições, elas foram recebidas em sequência sem a necessidade de monitoramento do usuário e todas as requisições foram recebidas.

O terceiro teste foi realizado utilizando o botão Disparador Bluetooth. O mesmo foi pareado com o notebook e foram realizados 100 testes. Novamente, a taxa de entrega foi de 100%.

O quarto e quinto testes foram utilizando o notebook como tcpServidor e o *desktop* como tcpCliente. Foram testados o sistema com e sem interface. O teste com interface foi parecido com o da primeira vez: o tcpServidor abre apenas uma interface por vez, porém, devido ao fato de o notebook utilizar o sistema operacional Linux Mint 20.1, ele foi capaz de armazenar quatro outras requisições além daquela que ele abriu a interface, dessa forma, o “travamento” que ocorria no Windows, quase não era percebido no Linux, mas era necessário um usuário fechando as telas que fossem abertas para que o fluxo de requisições também fosse contínuo. O teste sem a interface foi igual ao Windows, as requisições chegaram sem “travamento”. Nos dois testes todas as requisições foram recebidas.

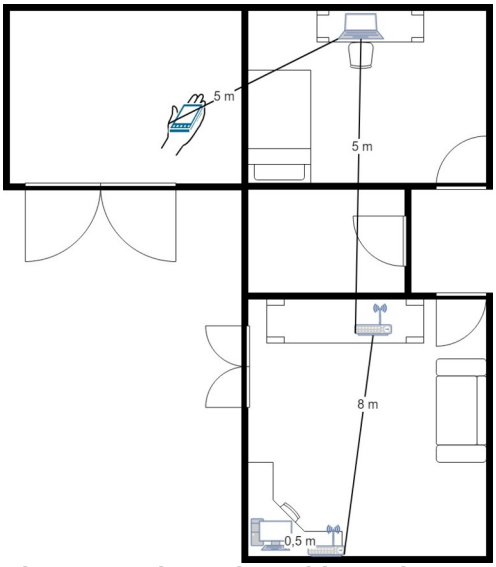

Figura 13: Planta do ambiente de teste 1 ao 5

O sexto teste foi realizado mudando a localização do notebook, afastando ele um pouco mais do roteador, como mostrado na Figura 14. O teste realizado foi utilizando o notebook como tcpCliente, o computador como tcpServidor e foi utilizado a interface gráfica do tcpServidor. Por mais que a distância tenha sido alterada, não houve mudança no resultado, 100% das requisições chegaram ao destino.

O sétimo teste o tcpServidor foi executado no notebook e o tcpCliente no computador. O resultado continuou o mesmo. O oitavo teste foi utilizando o Disparador Bluetooth, foi mudado sua posição para que ficasse mais distante do notebook. Novamente foram realizados cem testes e os cem obtiveram sucesso.

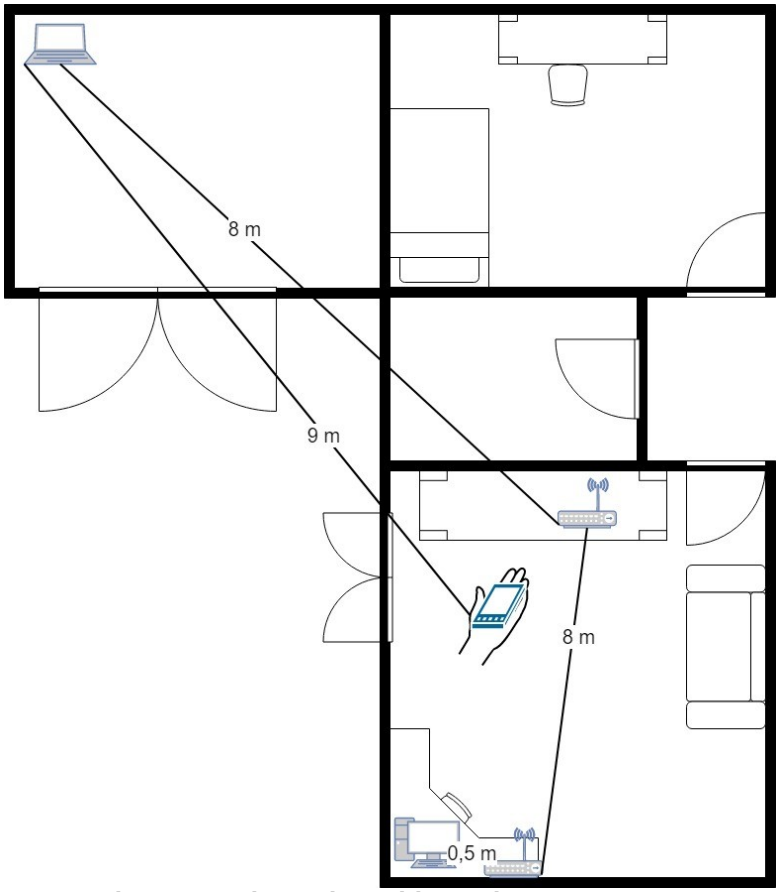

Figura 14: Planta do ambiente de testes 6 ao 8

Para o teste 9, 10 e 11 foi mudado a conexão Wi-Fi do notebook para o roteador mais próximo do computador, como mostrado na Figura 15. Esse era o teste de maior estresse que poderia ser feito neste ambiente de testes, mas não houve mudança no resultado. O teste 9 foi configurado como o notebook sendo o tcpCliente e o computador o tcpServidor, o teste 10 o notebook virou o tcpServidor e o computador o tcpCliente e por último o teste 11 foi realizado com o Disparador Bluetooth.

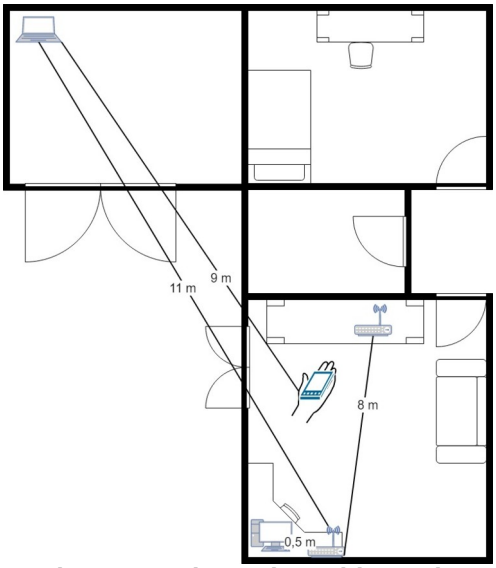

Figura 15: Planta do ambiente de testes 9 a 11

Como forma de consolidar os testes, foi realizado mais 3 testes onde o *laptop* voltou para o quarto e os testes foram realizados novamente, como apresentado na Figura 16. O teste 12 foi realizado com o *laptop* sendo tcpCliente e o *desktop* sendo tcpServidor. No teste 13, o *laptop* se tornou o tcpServidor e o *desktop* o tcpCliente. Já no teste 14 foi utilizado o disparador bluetooth. Como resultado, o sistema respondeu a 100% das entregas.

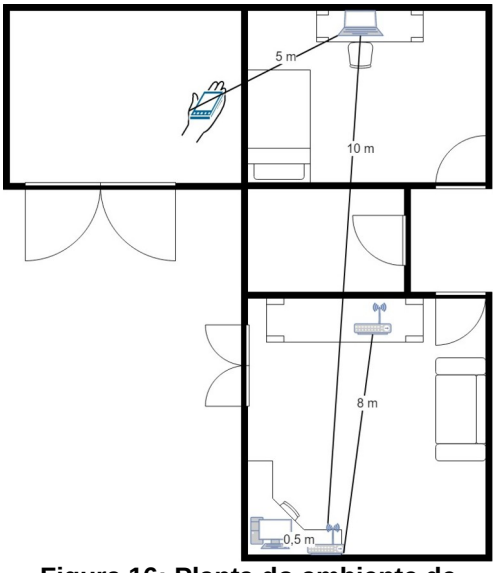

Figura 16: Planta do ambiente de testes 12 a 14

8. Conclusão

O presente trabalho apresentou o desenvolvimento de um sistema antipânico com foco principal em hospitais públicos, mas que pode ser utilizado em outras circunstancias, como hospitais/clínicas veterinárias, escolas, dentre outros lugares onde há a

necessidade de pedir ajuda de forma discreta e objetiva.

Foi desenvolvido os componentes de comunicação “cliente” (consultórios) e “servidor” (líderes hospitalares) usando o protocolo TCP, além de bancos de dados e interfaces GUI para a manutenção do sistema. Além disso, a solução foi planejada para ter baixa latência na entrega dos pacotes, desenvolvida com tecnologias gratuita, de código aberto. Foi desenvolvido um componente que trabalha com Disparadores Bluetooth, para o chamado de ajuda quando o funcionário estiver longe do computador e um sistema de salvamento de ocorrências para estudos posteriores ao socorro.

Os resultados dos testes obtiveram sucesso, embora não implementado em nenhuma unidade de saúde para avaliar o sistema em produção. Então, como sugestão de trabalhos futuros, a implementação e avaliação do sistema em ambiente de produção é recomendada, além de testes de usabilidade das interfaces GUI e alteração da disposição dos objetos na tela. Implementações da solução utilizando outras tecnologias também são bem-vindas. Outras sugestões de trabalhos futuros são o teste do sistema utilizando o protocolo UDP, a fim de comparar o desempenho do mesmo em relação a abordagem atual (TCP). Além disso, para suportar mais interfaces abertas ao mesmo tempo, pode-se dividir as requisições que chegam no tcpServidor em *threads*. Também é bem-vinda a implementação do botão *bluetooth* através de um dispositivo apropriado, para que não haja a necessidade de um pareamento prévio com o dispositivo para seu acionamento. Por fim, também é necessário realizar mais testes com o sistema, sobretudo em situações de estresse na rede.

O presente trabalho utilizou o conhecimento de várias disciplinas do Curso Superior de Análise e Desenvolvimento de Sistemas, como a matéria de Redes de Computadores, onde foi ministrado sobre os protocolos TCP e UDP, além de aulas práticas como desenvolvimento de um sistema de comunicação em rede utilizando *Socketes*, como usado na proposta desse trabalho. Outras aulas importantes foram as de banco de dados, onde foi ensinado sobre a normalização, o que ajudou a desenvolver tabelas mais objetivas para o trabalho. Também as aulas de Arquitetura de Software e a Metodologias Ágeis, que ensinaram sobre desenvolvimento de diagramas e gerenciamento de projetos ágeis, o que ajudou na criação das *Sprint*s. Por último, as aulas de Java e Programação Orientada a Objetos, que mostraram na prática como desenvolver sistemas orientados a objetos.

Além dos ensinamentos aprendidos no Curso, também foram adquiridos vários conhecimentos extras, como conhecimento sobre a linguagem Python, criação do banco de dados utilizando uma tecnologia ORM e conhecimento sobre métodos de usar uma tecnologia Bluetooth para trabalhar em conjunto com o software.

Referências

Brinkman, J. (2006). *Alarm over IP een Feit*

DE TRABALHADORES DE ENFERMAGEM. *21*, 1024. https://doi.org/10.5935/1415-2762.20170034

GTK. (2021). *The GTK Project - A free and open-source cross-platform widget toolkit*. https://www.gtk.org/docs/installations/windows

Kurose, J. F., & Ross, K. W. (2013). Redes de Computadores e a Internet. In *Editora PEARSON*.

Neto, C. C., & Raposo, L. D. O. (2014). *SMTP x POP3, TCP X UDP, FTP, HTTP*.

OpenGL. (2021). *OpenGL Overview*. https://www.opengl.org/about/

Pai, D. D., Lautert, L., Souza, S. B. C. de, Marziale, M. H. P., & Tavares, J. P. (2015). Violence, *Burnout* and Minor Psychiatric Disorders in Hospital Work. *Revista Da Escola de Enfermagem Da USP*, *49*(3), 457–464. https://doi.org/10.1590/S0080-623420150000300014

Python. (2021a). *Interfaces Gráficas de Usuário com Tk — documentação Python 3.9.5*. https://docs.python.org/pt-br/3/library/tk.html

Python. (2021b). *Python*. python.org

Security, C. R. (2021). *Emergency Call Systems - Security Solutions - High security booths, portals, doors, locks.* https://www.cometaspa.com/en/Solutions/Emergency-Call-Systems/

Silveira, J., Karino, M. E., Martins, J. T., José, M., Galdino, Q., & Schmitt Trevisan, G. (2014). *Violência no trabalho e medidas de autoproteção: concepção de uma equipe de enfermagem*.

Sockets Linux. (2008). https://books.google.com.br/books?hl=pt-BR&lr=&id=pM-bdTCkS8gC&oi=fnd&pg=PA3&dq=portas+x+sockets&ots=5-dh0p5hdi&sig=uQX1nD0k6XgTG6XDbibAmIdltdw#v=onepage&q=portas&f=false

SQLAlchemy. (2021). *SQLAlchemy - The Database Toolkit for Python.* https://www.sqlalchemy.org/

SQLite. (2021). *Sobre SQLite*. https://www.sqlite.org/about.html

Wazlawick, R. (2013). *Engenharia de software: Conceitos e práticas*.

Windows. (2021). *O DNS funciona em TCP e UDP - Windows Server | Microsoft Docs*. https://docs.microsoft.com/pt-br/troubleshoot/windows-server/networking/dns-works-on-tcp-and-udp

Farina, Ivan. (2021). Botão Pânico para hospitais públicos. https://github.com/ivanfarina08/Botao_Panico_para_hospitais_publicos

Documento Digitalizado Restrito

Artigo Final - Ivan Farina

Assunto: Artigo Final - Ivan Farina
Assinado por: Rodolfo Oliveira
Tipo do Documento: Outro
Situação: Finalizado
Nível de Acesso: Restrito
Hipótese Legal: Direito Autoral - conservar a obra inédita (Art. 24, III, da Lei nº 9.610/1998)
Tipo do Conferência: Documento Original

Documento assinado eletronicamente por:

- **Rodolfo Francisco de Oliveira**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 04/01/2022 17:16:29.

Este documento foi armazenado no SUAP em 04/01/2022. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 859560
Código de Autenticação: d432c7b9a7