Remotto: Relatório Técnico

Discente: Lucas Henrique Vilela
Orientador: Eliseu César Miguel

February 2023

1 Apresentação

Remotto é um sistema de controle remoto para sistemas computacionais construído sobre redes locais sem fio. Ele executa ações comuns do uso de computadores como, movimentação do ponteiro do *mouse*, digitação de texto e acesso a atalhos previamente definidos, sejam eles funções do Sistema Operacional (SO) ou combinação de teclas.

A falta de opções de qualidade em produtos similares disponíveis no mercado foi a grande motivação para o desenvolvimento deste trabalho. Em sua maioria, os produtos apresentam falhas com relação à confiabilidade, uso excessivo de propagandas ou má qualidade no geral. Outra motivação importante é oferecer uma base para o futuro desenvolvimento de aplicativos que possibilitem a usabilidade aos sistemas por pessoas com dificuldades motoras ao controle de um computador.

2 Introdução

Ao se buscar meios que permitam o controle remoto de um computador pessoal, várias alternativas podem ser encontradas. Desde *hardware* como teclados sem fio[1] até a possibilidade do acesso remoto, onde a imagem da tela do computador é espelhada no dispositivo [2]. Existem também aplicativos móveis que se propõem a solucionar este problema[3]. Em sua maioria, eles são disponibilizados de forma gratuita em lojas de aplicativos, como a Play Store e cumprem seu objetivo. Por outro lado, como é observado com frequência nessas plataformas, o objetivo principal desses aplicativos não é a entrega da solução e sim o lucro por meio da exibição de anúncios, da venda de "versões *premium*" e em alguns casos a coleta de informações do usuário.

[1] `https://www.techtudo.com.br/review/mini-teclado-sem-fio-clone.ghtml`

[2] `https://olhardigital.com.br/2020/03/25/dicas-e-tutoriais/as-5-melhores-ferramentas-de-acesso-remoto-para-usar-no-pc/`

[3] `https://olhardigital.com.br/2018/09/14/dicas-e-tutoriais/como-usar-o-smartphone-de-controle-remoto-para-o-computador/`

O objetivo deste trabalho é criar um sistema de *software* que permita ao usuário controlar as funcionalidades de um computador a partir de um dispositivo móvel por meio de conexão de rede de computadores via *Wireless Fidelity* (WI-FI). O público alvo do sistema são usuários comuns de computadores pessoais.

Como resultado, foi desenvolvido um produto que oferece ao usuário a possibilidade de controlar um computador pessoal por meio de aplicativo móvel, permitindo, por exemplo, a digitação de texto e o controle do *mouse*. O sistema possui código aberto, o que permite auditoria, modificação, compilação de forma independente e, também, instalação de forma gratuita por qualquer usuário.

O restante deste documento está organizado da seguinte forma: A Seção "Revisão Técnica" lista aplicativos com funcionamento semelhante ao do Remotto. A Seção "Informações Técnicas" traz informações sobre a licença escolhida, os requisitos para implementação e os Sistemas Operacionais onde o produto foi testado. A Seção "Desafios e benefícios do Produto" discute de modo geral os desafios encontrados durante o desenvolvimento e os benefícios dele para o público. A Seção "Funcionalidades" trata de forma individualizada das funções presentes no produto. A Seção "Arquitetura" apresenta de forma detalhada a implementação do Cliente, do Servidor e do Protocolo de comunicação utilizado. Finalmente, a Seção "Desafios Multiplataforma" discute os desafios encontrados para manter a compatibilidade entre diferentes Sistemas Operacionais.

3 Revisão técnica

Existem vários aplicativos que trazem funcionalidades semelhantes às do Remotto. As diferenças podem ser notadas no design da interface, nas funcionalidades, no servidor e na licença. A seguir apresentamos 3 desses aplicativos.

3.1 *Remote Control Collection*

O aplicativo *Remote Control Collection* [1] possui código fechado e monetização através de propagandas. Apresenta um grande número de controles e funções aglomeradas em diversas telas por onde o usuário pode navegar. O servidor possui código aberto porém tem uma licença restritiva que dificulta a colaboração. Ele tem suporte ao Microsoft® Windows® e ao Mac OSX e não recebe atualizações desde 2015. As avaliações do aplicativo são ligeiramente positivas apesar de ser comum encontrar reclamações de usuários sobre vários aspectos.

3.2 *Unified Remote*

Unified Remote [2] também possui código fechado e conta com monetização através de propagandas. Um diferencial é a presença de funções específicas para programas do computador além dos controles genéricos. O servidor possui código fechado e ele oferece suporte a várias plataformas. Possui avaliações positivas.

3.3 PC remoto

Diferentemente dos anteriores, PC remoto [3] oferece a possibilidade do espelhamento da tela e áudio do *smartphone* no computador assim como o contrário. Ele também traz controles gerais como os demais aplicativos e a monetização utilizando propagandas. O código é fechado e as avaliações são medianas, sendo possível encontrar diversas reclamações acerca de funcionalidades e a conexão.

4 Informações Técnicas

Em busca de oferecer um produto democrático aos usuários, optou-se pela licença Apache 2.0[4], considerando o perfil fortemente aberto, e tão importante, por não exigir limitações na utilização do *software* como acontece com outras licenças, como a licença GPL[5]. Também, levou-se em consideração o uso da licença MIT[6], mas esta foi deixada em favor da Apache 2.0, pois a última garante direitos sobre o nome da aplicação, elemento importante para fins de registro e proteção contra cópias maliciosas.

O sistema tem requisitos de baixo nível de implementação e estes são divididos entre o cliente o servidor da seguinte forma:

4.1 Servidor

- Sistema Operacional: Microsoft® Windows® 7 ou mais recente; Sistemas baseados no Kernel Linux 4.4[7] ou mais recente com servidor gráfico X;
- *Java Runtime Environment*[8] 8 ou superior;
- CPU com 2 ou mais núcleos;
- 200 MiB de memória RAM;
- 50 MiB de espaço em disco;
- Placa de rede e Placa mãe com suporte a pacotes *Wake on Lan*[9] (opcional).

Para uso do Sistema Operacional Microsoft® Windows®, recomenda-se executar o servidor com permissão de administrador do sistema.

4.2 Cliente

- Android™ 5.1[10] ou superior;

[4] https://www.apache.org/licenses/LICENSE-2.0
[5] https://www.gnu.org/licenses/gpl-3.0.html
[6] https://mit-license.org/
[7] https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/ChangeLog-4.4.180
[8] https://www.java.com/en/download/help/release_changes.html
[9] https://forums.ivanti.com/s/article/Understanding-Wake-On-LAN?language=en_US
[10] https://pt.wikipedia.org/wiki/Android_Lollipop

- 30 MiB de espaço em disco;
- Permissão de acesso à rede;
- Permissão (opcional) de acesso à câmera do dispositivo;
- Permissão de controle da vibração do dispositivo.

Além disso, tanto o servidor quanto o cliente devem estar conectados à mesma rede *wireless*.

O sistema foi testado e executado com sucesso nos seguintes sistemas operacionais: Microsoft® Windows® 7, Microsoft® Windows® 10, Ubuntu 16.04 LTS, Ubuntu 18.04 LTS, Ubuntu 20.04 LTS, Ubuntu 22.04 LTS, Xubuntu 18.04 LTS, Xubuntu 22.04 LTS, Android 5.1, Android 8.1.

5 Desafios e benefícios do Produto

Existem alguns desafios a serem superados, sobretudo em relação à compatibilidade entre sistemas, a latência de comunicação e abertura para a expansão do produto.

Embora os sistemas operacionais, como o Microsoft® Windows® e o Ubuntu, apresentem certa similaridade, algumas peculiaridades fazem com que mecanismos de controle precisem ser implementados para cada situação. Isto é, realizar a chamada do método *System.getProperty("os.name")* para identificar em qual Sistema Operacional o código está sendo executado e a partir disso escolher o bloco de código compatível. Um exemplo disso é a funcionalidade de suspender as atividades do dispositivo pareado. A implementação é feita por meio da chamada de comando que no Microsoft® Windows® segue a sintaxe do *Command Prompt* e no Ubuntu, o *Bash*. Com isso, apesar da *Java Virtual Machine* (JVM) oferecer todo o suporte necessário para a escrita de *software* multiplataforma, ainda é necessário este tipo de ajuste fino para que de fato tudo funcione.

Outro ponto extremamente importante é a latência da comunicação entre cliente e servidor. Embora seja tolerável um certo nível de atraso para tarefas de digitação, o mesmo não pode ser dito para o uso do *mouse*. A característica principal deste tipo de dispositivo é transferir, com fidelidade, o movimento do dispositivo gerado pelo usuário para a tela do computador. Caso ocorra um atraso perceptível entre a ação do usuário e a animação que ocorre na tela, o uso do recurso se torna estressante e improdutivo. Tudo isso precisa ser observado em dispositivos cabeados e torna-se ainda mais desafiador quando tratamos de redes sem fio, onde inúmeros fatores podem atrasar a entrega de pacotes ou mesmo causar a perda deles. Para poder levar à melhor experiência possível, foi necessário fazer um cuidadoso equilíbrio entre o uso de *Threads* para não sobrecarregar o processador do *smartphone* e do computador pessoal, fixar o tamanho dos pacotes e também controlar o número de pacotes enviados para não sobrecarregar a rede *wireless*.

Embora o produto inicialmente esteja restrito ao controle do *mouse* e do teclado de um computador pessoal usando recursos disponíveis em *smartphones*,

desde o início foi necessário levar em conta a possibilidade da expansão de suas funcionalidades. Partindo desse ponto, o formato de mensagens proposto foi o primeiro passo do projeto. A combinação dele com o *User Datagram Protocol* (UDP) [4] torna possível a criação de novas versões do cliente e do servidor, utilizando outros recursos dos *smartphones* (por exemplo, câmeras e acelerômetro), outras linguagens, outros dispositivos como *Internet of things* (IoT)[11], e o mais importante: a sua interoperabilidade. Assim, um terceiro pode desenvolver um cliente feito para o IOS[12] da Apple e ainda ser totalmente compatível com o servidor apresentado. Da mesma forma, também é possível desenvolver um servidor específico para ser executado em uma *Smart TV* e que continue compatível com clientes já desenvolvidos.

6 Funcionalidades

De uma maneira geral, o sistema desenvolvido fornece acesso a basicamente dois dispositivos de entrada e saída comuns aos computadores pessoais: o teclado e o *mouse*. Com o acesso a estes dispositivos, comandos fornecidos no dispositivo móvel são repassados ao computador pessoal, que os interpreta como se fossem fornecidos diretamente nos dispositivos de entrada e saída locais. Com isso, pode-se, por exemplo, digitar um texto em um editor de textos do computador a partir do dispositivo móvel.

Toda a interação acontece por meio do cliente para Android que está dividido em 3 janelas de interface principais, apresentadas a seguir, além das janelas de configurações.

6.1 Tela *Home*

Ao abrir o cliente o usuário se depara com a Tela *Home*, que é representada na Figura 1. Nesta tela, tem-se acesso a uma interface que simula um controle remoto de TV. Nesta tela ele pode controlar elementos básicos como volume do dispositivo, avançar ou retroceder faixas de mídia em execução assim como controlar seu status (pausar, parar ou reproduzir). Também existem nesta tela, botões de navegação, o botão especial "*Enter*", e dois botões de controle do dispositivo para ligá-lo ou colocá-lo em estado suspenso. Para que ele possa ser ligado, a placa de rede do dispositivo precisar ter suporte necessário ao sistema e ser configurada para reagir a pacotes *Wake on Lan*. Existe também um botão intitulado "Mais" que abre uma janela flutuante com mais dois botões, sendo eles: "Mais zoom" e "Menos zoom". A função deles é controlar o nível de zoom no dispositivo pareado.

Como o Android oferece uma vasta configuração de tamanhos de tela e resoluções, é prudente adicionar os botões de zoom nessa área separada para fazer melhor uso do espaço. Isso também permite que no futuro novas opções possam

[11] `https://en.wikipedia.org/wiki/Internet_of_things`

[12] Sistema Operacional para dispositivos móveis. Mais informações podem ser encontradas em: `https://pt.wikipedia.org/wiki/IOS`

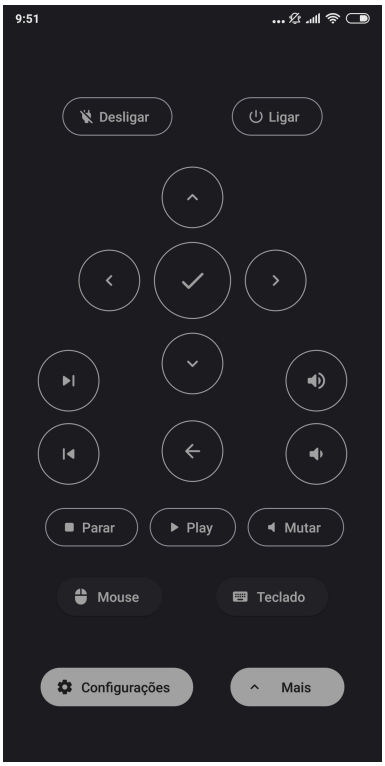

Figura 1: Tela *Home*

ser adicionadas sem a necessidade de se fazer grandes rearranjos no *layout* principal. Por fim, a Tela *Home* possui mais 3 botões que levam o usuário às outras telas do cliente, sendo eles: O botão "*Mouse*", "Teclado" e "Configurações".

6.2 Tela *Mouse*

Ao tocar no botão "*mouse*", o usuário é direcionado a uma tela vazia, com fundo escuro e com brilho reduzido e que se mantém ativa ao longo do tempo, mesmo quando não existe interação. Esta é a Tela *Mouse*, representada pela Figura 2. O propósito desta tela é agir como um Touchpad presente em *notebooks* e permitir o controle do *mouse* do dispositivo.

O uso desta tela também é semelhante ao visto em *notebooks* funcionando com gestos padrões onde:

- O movimento de um dedo na tela é repassado ao ponteiro do *mouse* no dispositivo pareado;
- A velocidade com que se move o dedo é traduzida em uma aceleração do movimento da seta do *mouse*, fazendo com que ela se desloque por uma

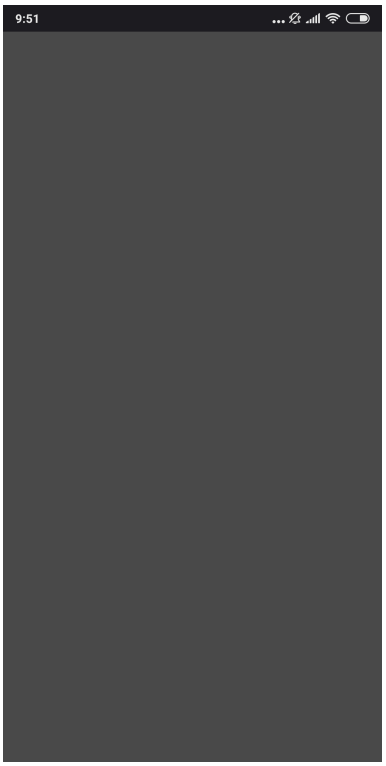

Figura 2: Tela *Mouse*

área maior da tela;

- Um toque na tela com um dedo é entendido como um clique no botão esquerdo do *mouse*.
- O toque duplo na tela com um dedo em um curto espaço de tempo é entendido como um clique duplo no botão esquerdo do *mouse*;
- O toque com dois dedos na tela é entendido como clique no botão direito do *mouse*;
- Ao se deslizar com dois dedos sobre a tela na vertical é simulado o botão de *scroll* do *mouse* para cima e para baixo;
- O toque rápido da tela seguido de um novo toque onde se mantém o contato é entendido como a ação de pressionar o botão esquerdo do *mouse*.

A cor escura escolhida para o fundo desta tela deve-se ao fato de que alguns dispositivos possuem telas que sofrem de um efeito conhecido como *burn in*, onde elementos muito claros ficam "fixos" como se marcassem a tela em caso

de exibição por longo período de tempo. Já o brilho baixo faz sentido, pois esta tela em específico não precisa ser "vista" pelo usuário. A sua atenção estará totalmente voltada ao dispositivo que está operando através do cliente. Assim é evitado o desperdício de energia e problemas na tela por longos períodos de utilização.

6.3 Tela Teclado

A partir da Tela *Home*, ao tocar no botão Teclado, o usuário é direcionado a uma tela onde ele pode digitar texto. Esta tela é representada pela Figura 3. Como é possível notar, esta tela dispõe de um Touchpad como descrito em "Tela *Mouse*", botões representando teclas muito utilizadas durante digitação como: Espaço; *Enter*; Apagar; teclas de atalho rápido, como Copiar e Colar; e por fim uma caixa de texto.

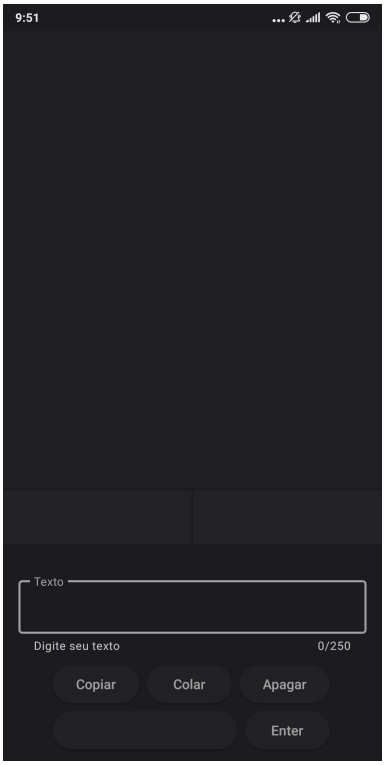

Figura 3: Tela Teclado

Todos os botões realizam ações padrões como sugeridas pelos nomes dos próprios botões e, ao interagir com a caixa de texto, abre-se o teclado virtual do aparelho, como é possivel ver na Figura 4. O texto digitado é exibido ao usuário até o limite de 250 caracteres e ao tocar no botão *Enter* ele é enviado

ao dispositivo pareado. Essa tela não possui o recurso de baixo nível de brilho descrito na Tela *Mouse* pois diferente dela, esta tela precisa de atenção do usuário enquanto este interage com ela. Também não conta com o recurso de permanecer ativa sempre, pois não é necessário. Por fim, o *Touchpad* desta tela conta com dois botões virtuais simulando o botão esquerdo e direito presente em *mouses*. Apesar da área do *Touchpad* contar com os mesmos gestos presentes na "Tela *Mouse*", por ser uma tela focada na digitação de texto é mais conveniente ao usuário ter sempre a sua disposição botões para acelerar o trabalho.

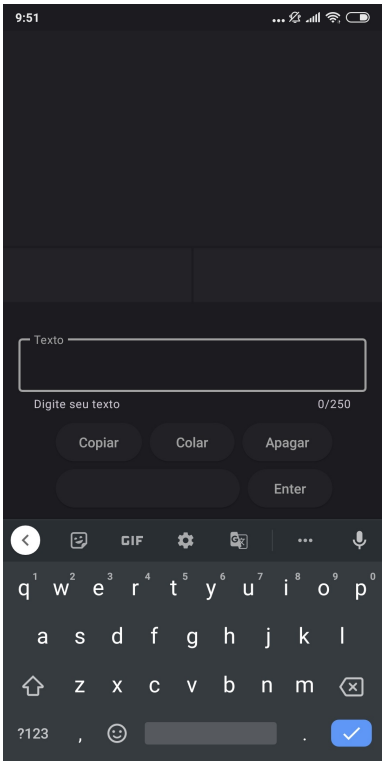

Figura 4: Tela Teclado - Com o teclado virtual

O limite de 250 caracteres pode sofrer mudanças de acordo com o texto digitado pelo usuário. Ele vem da especificação do padrão de mensagens para comunicação do cliente com o servidor combinada com a especificação da codificação *8-bit Unicode Transformation Format* (UTF-8) [5]. O tamanho dos pacotes é fixado em 256 *bytes* e o UTF-8 utiliza "um *byte* para codificar os 128 caracteres do sistema de representação *American Standard Code for Information Interchange* (ASCII) [6] (*Unicode* [7] U+0000 a U+007F)", "dois *bytes* para caracteres Latinos com diacríticos" e três ou quatro *bytes* para o restante dos caracteres. O limite foi imposto levando-se em conta que boa parte dos caracteres digitados ocuparão apenas 1 *byte*. Por isso, caso o usuário utilize um ou

mais caracteres que necessitem de mais de um *byte* para serem representados, o limite de caracteres do texto será decrescido de acordo.

6.4 Interface gráfica do servidor

O servidor possui uma interface gráfica simplificada que permite o seu gerenciamento. Como é possível observar na Figura 5, a janela principal conta com um Código QR[13] usado para facilitar o pareamento com o cliente. Ela também conta com a lista de clientes atualmente pareados e a informação do endereço *Internet Protocol* (IP)[14] e *Media Access Control* (MAC)[15] do computador onde o servidor está sendo executado. Na parte superior existem controles para Pausar e Retomar a execução do servidor e também controles para bloquear ou permitir um cliente pareado.

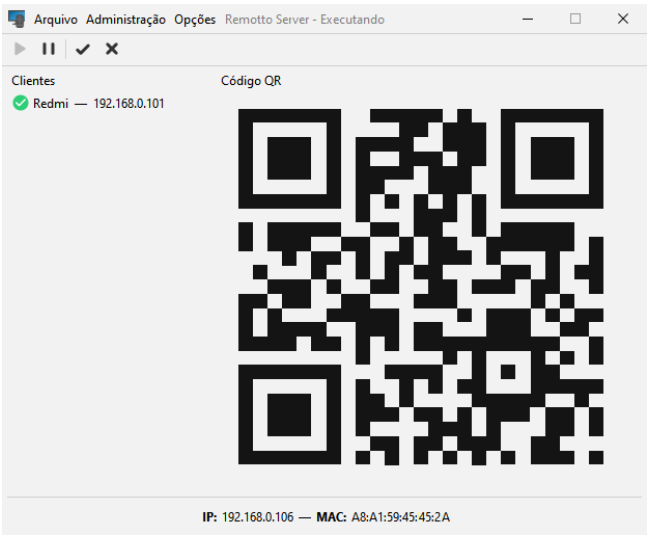

Figura 5: Interface gráfica do servidor

7 Arquitetura

O sistema é baseado em 3 pontos principais:

1. O servidor, que é um *software* que deve ser executado na máquina (computador de uso pessoal) que se deseja operar à distância;

[13] `https://en.wikipedia.org/wiki/QR_code`
[14] `https://en.wikipedia.org/wiki/IP_address`
[15] `https://en.wikipedia.org/wiki/MAC_address`

2. O cliente, que após ser pareado ao servidor, funciona como interface de controle que permite ao usuário executar ações na máquina cujo servidor está instalado.

3. O roteador *wireless* fornece uma rede local onde o cliente pode se comunicar com o servidor;

A Figura 6 ilustra a arquitetura e indica o fluxo de dados.

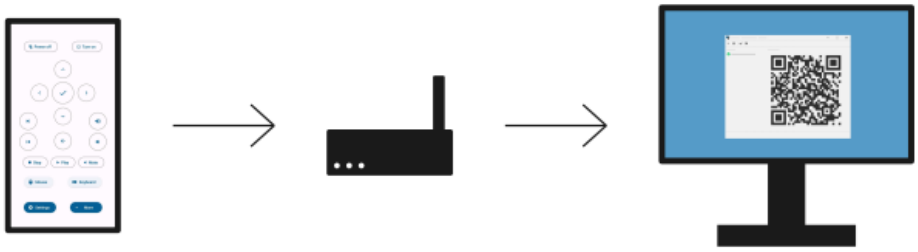

Figura 6: Representação da comunicação

7.1 Protocolo de comunicação

A comunicação entre cliente e servidor é feita através de pacotes UDP com mensagens formatadas. Os pacotes são enviados exclusivamente do cliente e recebidos pelo servidor. Deste modo, não existe nenhum tipo de comunicação do servidor para o cliente, apenas do cliente para o servidor. Isso facilita o desenvolvimento de ambos pois não é necessário estabelecer uma conexão real entre os dois. As mensagens são formatadas de acordo com o padrão apresentado a seguir.

Uma mensagem terá o tamanho máximo de 256 *bytes*. Ela segue a estrutura a seguir onde os caracteres X indicam o tipo da ação, os caracteres & identificam o cliente e os caracteres @ indicam dados adicionais que podem ser enviados:

XXX&&&&&&&&&@@@@@@@@@@@@@@@@@@@@@@@@@@ ... @

Os caracteres de índice de 0 a 2 são uma combinação de letras e números [0-9][a-z] destinada a identificar um tipo de ação. Por questão de organização, as ações de um mesmo tipo são agrupadas. Para isso o caractere de índice 0 é fixo enquanto os dois seguintes variam, porém caso necessário novas ações podem ser adicionadas sem necessariamente seguir nenhum padrão.

Os grupos de ações disponíveis são:

- K: identifica uma ação de pressionar tecla;
- M: identifica uma ação relacionada ao controle do *mouse*;
- C: identifica uma ação de comando complexo controlado pelo servidor;
- T: identifica uma ação de escrita de texto;
- S: identifica uma ação especial.

Com isso as aplicações que implementam o protocolo devem possuir uma tabela de mapeamento (geralmente usando um enum) seguindo o esquema indicado na Tabela 1.

Grupo	Constante	Valor	Descrição
T	TEXT_WRITE	T00	Envia texto do cliente para o servidor.
C	ZOOM_IN	C01	Aumenta o nível de zoom no servidor.
C	ZOOM_OUT	C02	Diminui o nível de zoom no servidor.
C	COPY	C03	Copia o texto selecionado no servidor para a área de transferência do servidor.
C	PASTE	C04	Cola o conteúdo da área de transferência do servidor.
M	LEFT_CLICK	M01	Clique com o botão esquerdo do *mouse*.
M	RIGHT_CLICK	M02	Clique com o botão direito do *mouse*.
M	LEFT_DOUBLE_CLICK	M03	Duplo clique com o botão esquerdo do *mouse*.
M	MOVE	M04	Movimenta o cursor do *mouse*.
M	SCROLL_VER	M05	Rola o conteúdo na vertical.
M	SCROLL_HOR	M06	Rola o conteúdo na horizontal.
M	LEFT_HOLD_START	M07	Pressione o botão esquerdo do *mouse*.
M	LEFT_HOLD_END	M08	Libera o botão esquerdo do *mouse*.
S	SLEEP	S01	Coloca o servidor em estado suspenso.
S	SHUTDOWN	S02	Desliga o servidor.
K	PLAY	K01	Muda o estado da atual mídia entre reproduzindo e pausado.
K	STOP	K02	Para a atual mídia.
K	PREVIOUS	K03	Volta à mídia anterior.
K	NEXT	K04	Avança para a próxima mídia.
K	VOL_DOWN	K05	Diminui o volume.
K	VOL_UP	K06	Aumenta o volume.
K	MUTE	K07	Desliga o som.
K	ENTER	K08	Pressiona a tecla enter.
K	UP	K09	Pressiona a tecla seta para cima.
K	DOWN	K10	Pressiona a tecla seta para baixo.
K	LEFT	K11	Pressiona a tecla seta para esquerda.
K	RIGHT	K12	Pressiona a tecla seta para direita.
K	BACKSPACE	K13	Pressiona a tecla apagar.

Tabela 1: Tabela de mapeamento de constantes

Os caracteres de índice 3 até 11 são usados para identificação do cliente que está enviando a mensagem. Os clientes devem possuir preferencialmente nomes únicos contendo apenas letras, números e/ou os caracteres [“.”,“_”,“-”] para o correto controle de acesso no servidor. Caso o nome tenha menos de 9 caracteres, os espaços restantes devem ser preenchidos pelo caractere “&” e serem removidos pelo servidor antes da exibição.

Algumas ações precisam de dados adicionais para serem executadas de forma correta. Por exemplo: A ação “T00” usada para enviar texto do cliente para o servidor precisa enviar o texto digitado pelo usuário. Para levar essas informações adicionais, ficam reservados os caracteres de índice 12 até 255. Eles devem ser interpretados pelo servidor de acordo com a ação especificada. No caso das ações que não precisam de informações adicionais o servidor deve descartar essa parte da mensagem.

7.1.1 Ações com dados adicionais e como interpretá-las

Enviar texto (T00): Os caracteres de índice 12, 13 e 14 devem ser interpretados como um número inteiro e ele representa o tamanho do texto enviado. Assim, o conteúdo da mensagem começa no caractere de índice 15 e vai até o tamanho especificado ou o máximo que é o índice 255.

Mover *mouse* (M04): Os caracteres de índice 12 até 15 devem ser interpretados como um número inteiro e ele representa a quantidade de pixels que o ponteiro do *mouse* deve ser movimentado no eixo X. Entende-se que números positivos indicam movimento para direita e negativos para esquerda. Caso o número seja negativo o índice 12 deve conter o caracter “-”.

Os caracteres de índice 16 até 19 devem ser interpretados como um número inteiro e ele representa a quantidade de pixels que o ponteiro do *mouse* deve ser movimentado no eixo Y. Entende-se que números positivos indicam movimento para baixo e negativos para cima. Caso o número seja negativo o índice 16 deve conter o caracter “-”.

Os caracteres de índice 20 até 31 devem ser interpretados como um número inteiro positivo e ele representa o número de sequência do movimento.

Rolagem vertical (M05): Os caracteres de índice 12 até 15 devem ser interpretados como um número inteiro e ele representa a quantidade de movimentos de rolagem que o *mouse* deve realizar no eixo Y. Entende-se que números positivos indicam movimento para baixo e negativos para cima. Caso o número seja negativo o índice 12 deve conter o caracter “-”.

Os caracteres de índice 16 até 28 devem ser interpretados como um número inteiro positivo e ele representa o número de sequência do movimento.

Rolagem horizontal (M06): Os caracteres de índice 12 até 15 devem ser interpretados como um número inteiro e ele representa a quantidade de movimentos de rolagem que o *mouse* deve realizar no eixo X. Entende-se que números

positivos indicam movimento para direita e negativos para esquerda. Caso o número seja negativo o índice 12 deve conter o caracter "-".

Os caracteres de índice 16 até 28 devem ser interpretados como um número inteiro positivo e ele representa o número de sequência do movimento.

7.1.2 Exemplos

Pressionar a tecla *enter*: Suponha que o nome do cliente seja "Red". A ação de pressionar uma tecla faz parte do grupo "K" e ao verificar a tabela é possível perceber que o *enter* é identificado pelo valor "08". Logo, o tipo da ação será "K08". Com essas informações teremos a seguinte mensagem:

K08Red&&&&&&&

Enviar texto: Suponha que o nome do cliente seja "Yellow". A ação de escrita de texto faz parte do grupo "T" e como é a única do grupo, o tipo da ação será "T00". Precisamos enviar o texto a ser escrito, o que caracteriza uma ação com dados adicionais. O texto será "Bom dia", que possui 7 caracteres. Com essas informações teremos a seguinte mensagem:

T00Yellow&&&007Bom dia

7.2 Servidor

O servidor é responsável por receber os comandos enviados pelo cliente, interpretá-los e executá-los corretamente. Ele é escrito na linguagem de programação Kotlin e compilado para o *bytecode* da *Java Virtual Machine*. Isso possibilita o uso de recursos modernos do Kotlin como os *Coroutines* [8] que são blocos de código assíncronos e também traz a facilidade de implantação da plataforma Java. Para reagir com eficiência aos pacotes e interações do usuário, o servidor utiliza 3 *Threads*, sendo duas fixas e uma terceira que existe apenas quando o servidor não está ocioso. Além delas, também utiliza as *Threads* comuns da JVM como a *Thread* do coletor de lixo.

A primeira é a *Thread* principal onde é executada a interface de usuário. Ela é responsável por construir o *layout*, reagir a eventos provocados pelo usuário e controlar o acesso aos recursos.

A segunda *Thread* é onde a repetição responsável pelo recebimento de pacotes é executada. Dentro deste *loop* existe a chamada do método *receive()* da classe *DatagramSocket* [9] que aguarda o recebimento de pacotes UDP. É de extrema importância esta separação pois este método bloqueia a execução da *Thread* enquanto espera pela chegada de um pacote. Caso ele ficasse na principal a *Thread* ficaria congelada e seria impossível ao usuário fazer qualquer configuração no servidor em tempo de execução. Ao receber um pacote UDP o código é desbloqueado e volta ao seu fluxo normal.

Por fim, a terceira *Thread* é um caso especial. Ela é iniciada para tratar cada pacote UDP que chega ao servidor e finaliza após isso. Portanto, enquanto

o servidor está ocioso a *Thread* não é necessária. Enquanto o usuário utiliza as teclas virtuais ou a escrita de texto ela é constantemente iniciada e finalizada. No caso do *Touchpad*, o fluxo de pacotes é muito grande, então o número de *Threads* simultâneas é indefinida.

7.3 Cliente

O cliente é o meio pelo qual o usuário envia comandos para o servidor. Ele está disponível para a plataforma Android e é escrito na linguagem de programação Kotlin. Conforme explicado anteriormente, ao interagir com os botões presentes na interface, pacotes UDP com mensagens formatadas são enviados ao servidor. Isto é feito através da detecção do evento de clique fornecido de maneira abstraída pelo Android. Isso facilita bastante o desenvolvimento pois cada botão possui seu próprio evento, que ao ser acionado o trata da forma mais adequada. Na maioria dos casos basta invocar o método *sendPkg()* (presente na classe de controle de dados) que cuida da construção e envio do pacote.

A implementação do controle do *mouse* é mais complexa e precisa de acesso de baixo nível a *Application Programming Interface* (API) de toque da tela do dispositivo. O método *setOnClickListener*[16] que normalmente é usado em alto nível para lidar com eventos de toque na tela, não fornece informações suficientes. Por este motivo é necessário usar o método *setOnTouchListener*[17] que ao ser adicionado ao componente, passa a dar acesso a um evento que é chamado a cada interação com a tela. Este evento possui a posição em X e Y onde ele ocorreu além dos seguintes parâmetros:

1. **MotionEvent.ACTION_DOWN**: indica o início da interação com o componente (toque na tela);

2. **MotionEvent.ACTION_POINTER_DOWN**: indica que um ou mais toques foram detectados na tela após a ocorrência do primeiro evento e antes de sua finalização;

3. **MotionEvent.ACTION_MOVE**: indica um deslocamento do dedo do usuário sobre a tela. O evento é chamado repetidamente indicando este parâmetro a cada deslocamento, inclusive quando mais de um toque foi detectado;

4. **MotionEvent.ACTION_UP**: indica o fim da interação com o componente. Somente é ativado quando o usuário deixa de tocar na tela totalmente.

Todas essas variações de parâmetros ocorrem no bloco de código dentro do método *setOnTouchListener*. Isso dificulta seu tratamento pois é preciso fazer

[16] `https://developer.android.com/reference/android/view/View#setOnClickListener(android.view.View.OnClickListener)`

[17] `https://developer.android.com/reference/android/view/View#setOnTouchListener(android.view.View.OnTouchListener)`

um controle com base em estados passados e no tempo decorrido entre eventos. Outro ponto de dificuldade é a ocorrência de um novo evento que pode mudar a interpretação do estado atual. Apesar disso, com essas informações se torna possível simular o funcionamento de um *Touchpad*.

Os eventos são divididos em dois tipos: eventos de toque e eventos de movimento. Os eventos de toque levam em conta o tempo e o número de toques na tela para serem classificados como clique com o botão esquerdo, duplo clique com o botão esquerdo, clique com o botão direito, e a ação de "segurar". Os eventos de movimento como o nome sugere são baseados no deslocamento do dedo do usuário sobre a tela e são apenas dois: o movimento simples de um dedo e o movimento de dois dedos na vertical. A implementação segue a lógica de que cada um dos eventos é tratado de forma individual, porém todos compõem o estado da interação do usuário. Dessa forma, existe uma variável responsável por contar o número de interações, que é incrementada em 1 sempre que um evento ocorre e também um método chamado *postDelayed* que executa blocos de código com um *delay* predefinido[18].

Sempre que ocorre um evento existe uma verificação do seu tipo e do número de interações. O tempo para execução do método *postDelayed* é iniciado e caso nenhum novo evento ocorra, o código dentro do método é executado. Caso ocorra novos eventos, a incrementação da variável contadora de interações faz com que o código dentro do método *postDelayed* seja ignorado. Geralmente durante o evento **MotionEvent.ACTION_UP** (que normalmente indica o fim de uma interação) ocorre o *reset* das variáveis de controle.

8 Desafios Multiplataforma

A escolha da linguagem Kotlin compilada para a *Java Virtual Machine* para o desenvolvimento do servidor foi feita com base em dois fundamentos:

- O compartilhamento de código com o cliente;
- A possibilidade de distribuir o servidor para múltiplas plataformas tendo uma única base de código e compilação.

Porém, ao longo do desenvolvimento se tornou claro que por mais que o Java oferecesse um ambiente favorável ainda era preciso fazer muitos pontos de controle para que de fato o código pudesse ser executado em sistemas com arquiteturas distintas.

O obstáculo mais simples a ser superado é com relação a função de desligar / suspender as atividades do dispositivo pareado. O comando utilizado para realizar esta ação é único para cada Sistema Operacional. Isso nos obriga a sempre checar em qual SO o servidor esta sendo executado para utilizar o comando adequado.

[18] `https://developer.android.com/reference/android/os/Handler#postDelayed(java.lang.Runnable,long)`

O grande problema a ser solucionado é a simulação das teclas de mídia (Reproduzir / Pausar, Parar, Retroceder, Avançar, Volume, Mutar) do dispositivo pareado. Diferente das teclas comuns dos teclados que possuem mapeamento implementado na *Java Virtual Machine*, essas teclas precisam de uma implementação manual. Além disso, existe um outro fator: os códigos de identificação dessas teclas não são padronizados entre diferentes sistemas operacionais, nem mesmo quando consideramos apenas sistemas baseados no Kernel Linux. Para lidar com isso, é necessário escrever código na linguagem de programação C específico para cada sistema e em seguida disponibilizar sua chamada através da *Java Native Interface* (JNI)[19]. Para otimizar o tempo de desenvolvimento, ao invés de fazer esse processo manualmente foi decidido usar uma biblioteca que já implementou toda essa lógica, conhecida como *JNativeHook* [10]. Infelizmente, por conta da dificuldade de se obter e mapear as teclas corretamente nos mais variados sistemas, as teclas de mídia inicialmente funcionam apenas no Microsoft® Windows®. Existe uma atualização da biblioteca que trará a compatibilidade para sistemas Linux mas ela ainda não foi lançada.

Outro desafio a ser superado é a variedade de dispositivos que podem executar o sistema Android. Eles possuem especificações muito distintas, sobretudo em relação ao tamanho e densidade de pixels da tela. Isso faz com que um *layout* não funcione adequadamente em todos os dispositivos. Para superar isso, foi necessário a criação de variações que levam em conta o tamanho da tela. Este problema é conhecido e por isso o próprio Sistema Operacional cuida da escolha do melhor *layout*, bastando que o desenvolvedor forneça as variações. Um outro efeito da variação da densidade de pixels da tela é observado na utilização do recurso de *Touchpad*. A aceleração aplicada ao movimento gerado pelo usuário é menor em telas menores.

9 Créditos

As bibliotecas listadas a seguir foram integradas ao projeto e facilitaram muito o desenvolvimento.

- Darklaf - A themeable swing Look and Feel [11]
- JNativeHook - Global keyboard and mouse listeners for Java [10]
- QR Code generator library [12]
- Code scanner library for Android [13]

10 Conclusão

Neste relatório apresentamos o Remotto, que é um sistema de controle remoto para sistemas computacionais construído sobre redes locais sem fio. A motivação

[19] `https://docs.oracle.com/javase/8/docs/technotes/guides/jni/`

para o seu desenvolvimento veio da falta de opções de qualidade em produtos similares disponíveis no mercado e da possibilidade de servir de base para o futuro desenvolvimento de aplicativos que possibilitem a usabilidade aos sistemas por pessoas com dificuldades motoras ao controle de um computador.

O sistema já está pronto para uso com suporte do servidor ao Microsoft® Windows® e sistemas baseados no Kernel Linux. O cliente está disponível para a plataforma Android e esperamos que no futuro esse suporte se estenda a novos sistemas e plataformas.

Como proposta de trabalhos futuros recomendo o desenvolvimento de um cliente para a plataforma IOS. Sendo o segundo Sistema Operacional para dispositivos móveis com mais usuários é importante receber o suporte. Outro ponto importante é o desenvolvimento de clientes especiais que utilizando modelos de Inteligência Artificial e outras técnicas, possam oferecer acessibilidade a usuários com dificuldades motoras ao controle de um computador.

11 Agradecimentos

A Deus por me dar a capacitação necessária para desenvolver as atividades com facilidade. Aos meus pais e irmã, que me incentivaram nos momentos difíceis e deram todo o apoio necessário. Ao professor Eliseu, por ter sido meu orientador e ter desempenhado tal função com dedicação e amizade. Aos desenvolvedores das bibliotecas integradas neste projeto. A todos aqueles que contribuíram, de alguma forma, para a realização deste trabalho.

Referências

[1] S. Schultz, "Remote Control Collection: The Remote Control Collection allows you to remotely control your PC." `http://remote-control-collection.com/`, Acessado em 2023.

[2] P. Bergqvist and J. Berglund, "Unified Remote: The one-and-only remote for your computer." `https://www.unifiedremote.com/`, Acessado em 2023.

[3] Monect, "PC remoto." `https://www.monect.com/`, Acessado em 2023.

[4] J. Postel, "User Datagram Protocol." `https://www.rfc-editor.org/rfc/rfc768.txt`, Acessado em 2023.

[5] From Wikipedia, the free encyclopedia, "UTF-8." `https://en.wikipedia.org/wiki/UTF-8`, Acessado em 2023.

[6] From Wikipedia, the free encyclopedia, "ASCII, abbreviated from American Standard Code for Information Interchange." `https://en.wikipedia.org/wiki/ASCII`, Acessado em 2023.

[7] From Wikipedia, the free encyclopedia, "Unicode, formally The Unicode Standard." https://en.wikipedia.org/wiki/Unicode, Acessado em 2023.

[8] kotlinlang, "Coroutines: Asynchronous or non-blocking programming." https://kotlinlang.org/docs/coroutines-overview.html, Acessado em 2023.

[9] Oracle and/or its affiliates, "DatagramSocket: represents a socket for sending and receiving datagram packets." https://docs.oracle.com/javase/8/docs/api/java/net/DatagramSocket.html, Acessado em 2023.

[10] A. Barker, "JNativeHook: Global keyboard and mouse listeners for Java." https://github.com/kwhat/jnativehook, Acessado em 2023.

[11] J. Weis, "Darklaf: A themeable swing Look and Feel." https://github.com/weisJ/darklaf, Acessado em 2023.

[12] Project Nayuki, "QR Code generator library." https://github.com/nayuki/QR-Code-generator, Acessado em 2023.

[13] Y. Budiyev, "Code scanner library for Android." https://github.com/yuriy-budiyev/code-scanner, Acessado em 2023.