Aplicação Cliente-Servidor de

Urna Eletrônica Com Socket (TCP)

Danilo Guarnieri Cardoso N.USP 10442277, Gustavo Cesar Leite De Oliveira Santos N. USP 10442301, Murilo Pratavieira N. USP 8123082

Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo (USP)

{cardoso.nilo@gmail.com, gustavocesarlos@gmail.com, murilo.pratavieira@gmail.com}

Resumo. *Este artigo se trata de uma aplicação cliente-servidor criada em linguagens C e java com o intuito exclusivamente acadêmico. Para esse fim foi criado uma aplicação a fim de simular uma urna eletrônica. Onde o cliente irá fazer uma votação onde o servidor irá disponibilizar os candidatos e contabilizar o número de votos dos mesmos. Ademais, a aplicação contou com a utilização de sockets e threads, conceitos muito importantes no estudo de Redes de Computadores.*

1. Informações Gerais

A aplicação "Urna Eletrônica Com Socket (TCP)" aborda diversos temas importantes para área de Redes de Computadores. Neste projeto em específico foi abordado especialmente a camada de aplicação. Porém, também foi preciso um conhecimento sobre a camada de transporte.

O trabalho foi desenvolvido em três etapas:

1. **Construção do servidor utilizando linguagem Java**
2. **Construção do servidor utilizando linguagem C**
3. **Construção do cliente utilizando linguagem JavaFX**

Para construção dessa aplicação cliente/servidor também foi necessário a construção de um protocolo:

- **O carácter ',' separa os atributos de um mesmo candidato**
- **O carácter ';' separa um candidato do outro**
- **O carácter '!' indica o encerramento da mensagem - último carácter**
- **Os caracteres '999!' carrega a lista de candidatos do servidor**
- **Os caracteres '888!' prepara para o envio de votos da urna para o servidor e recebimento dos resultados (parciais ou finais) no cliente (urna)**

Além disso, a ordem a qual estipulamos para os atributos oriundos do servidor está na seguinte ordem: Código de Votação, Nome do Candidato, Partido e Número de Votos.

Sobre a comunicação do Cliente e Servidor, nossa aplicação está seguindo a seguinte ordem:

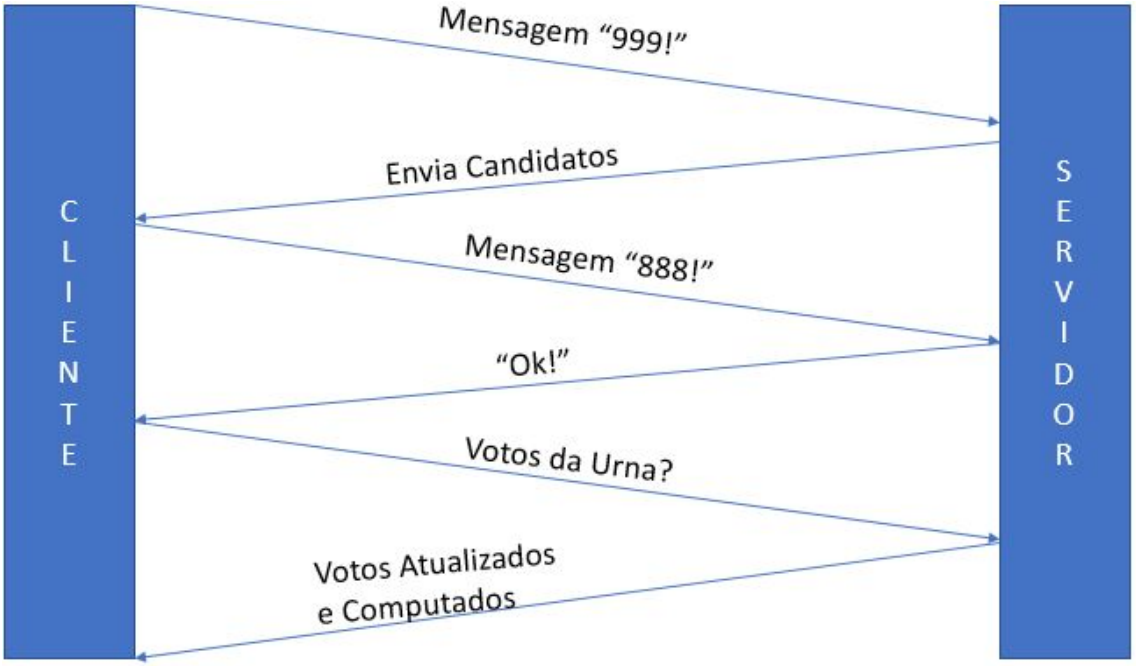

Figura 1. Mapeamento da Comunicação Cliente/Servidor

2. Fundamentação Teórica

Sobre as questões teóricas vale ressaltar quatro grandes tópicos:

2.1 - Conceito de Socket

O socket é um mecanismo comunicação entre máquinas. É utilizado na maior parte das vezes para implementar aplicações cliente/servidor, os quais permitem a troca de mensagens entre os processos de uma máquina/aplicação servidor e de uma máquina /aplicação cliente. Esse controle é feito pelo pelo Sistema Operacional (SO).

2.2 - Camada de Transporte

A camada de transporte, dentro do modelo OSI quanto no modelo TCP/IP, é a camada responsável pela transferência eficiente, confiável e econômica dos dados entre a máquina de origem e a de destino. Dentro dessa camada podemos citar o conceito do protoco TCP.

2.2.1 - Protocolo TCP

O protocolo TCP é um dos protocolos mais utilizados e conhecidos no campo de redes de computadores. Isto se deve ao fato dele garantir a entrega de todos os pacotes entre uma máquina emissora e outra receptora. Entre o estabelecimento desta ligação acontece um 'pré-acordo' que é denominado Three Way Handshake (SYN, SYN-ACK,

ACK).

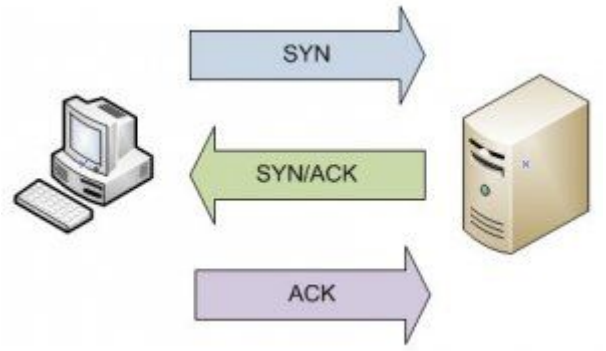

Figura 2. Three-Way Handshake

Além disso todas as conexões realizadas são *full-duplex* e *ponto a ponto*.

- ***Full-duplex:*** **Tráfego de dados podem ser feitos em ambas as direções ao mesmo tempo.**
- ***Ponto a Ponto:*** **Porque cada conexão possui exatamente dois pontos terminais únicos.**

2.3 - Camada de Redes

A camada de rede, outra camada de extrema importância, é a responsável por controlar a operação da rede de um modo geral. Suas principais funções são o roteamento dos pacotes entre a máquina fonte e a destino, o controle de congestionamento e a contabilização do número de pacotes ou bytes utilizados pelo usuário. O principal protocolo desta camada é o protocolo IP.

2.3.1 - Protocolo IP

O IP (Internet Protocol) é o principal protocolo de comunicação da Internet. Ele é o responsável por endereçar e encaminhar os pacotes que trafegam pela rede mundial de computadores e foi o utilizado na aplicação. Foi utilizado a versão 4 (IPv4), que possui 32 bits no campo de endereço.

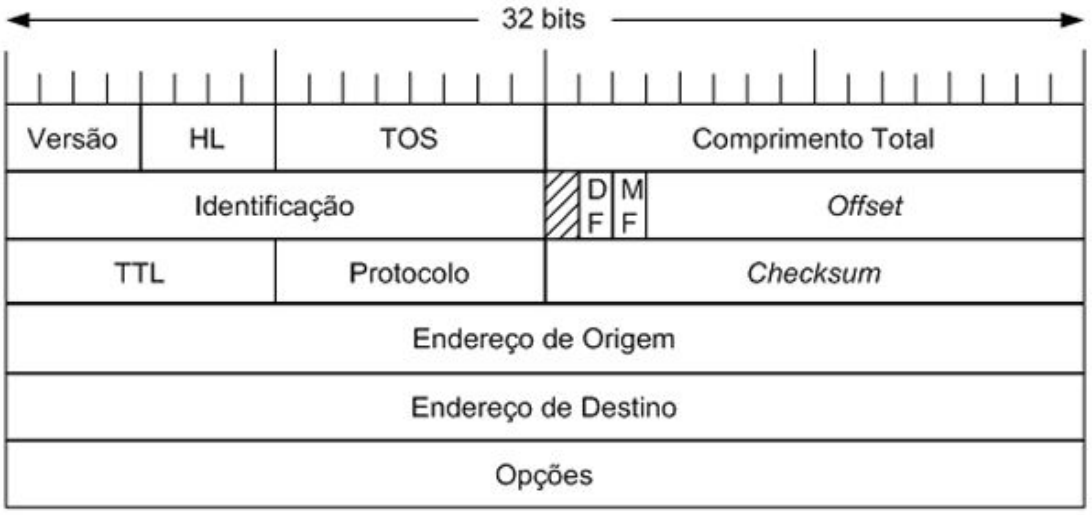

Figura 3. Cabeçalho de um Pacote IP

2.1 - Camada de Aplicação

Essa camada foi a mais evidente e trabalhada na nossa aplicação, uma vez que ela é responsável por prover serviços para aplicações de modo a separar a existência de comunicação em rede entre processos de diferentes computadores. Fizemos uma aplicação mais intuitiva possível a fim de facilitar o manuseio do usuário.

2.1 - Threads

Neste projeto foi utilizado threads no nosso servidor em java. Conseguimos abordar de maneira bem prática a utilização das threads a fim de fazer com que mais de um cliente (usuário) consiga fazer a votação na nossa urna eletrônica.

A Thread é um pequeno programa que trabalha como um subsistema, sendo uma forma de um processo se auto dividir em duas ou mais tarefas(neste nosso caso, servidores disponíveis para os clientes). Essas tarefas múltiplas podem ser executadas simultaneamente para rodar mais rápido do que um programa em um único bloco ou praticamente juntas, mas que são tão rápidas que parecem estar trabalhando em conjunto ao mesmo tempo.

3. Detalhamento da Solução

A aplicação desta urna eletrônica está dividida em três grandes partes:

3.1 Código Java - Cliente

Neste arquivo, temos duas principais classes, uma a qual irá conectar no servidor "*cosmos.laspdc.icmc.usp.br*" através da porta "*40003*", receber as mensagens oriundas do servidor e adquirir a mensagem de uma maneira que consiga identificar exatamente os dados. No cliente é realizado a codificação da mensagem para bytes a fim de enviar para o servidor para ser 'legível' tanto para o servidor codificado em linguagem java quanto para o codificado em linguagem C. O cliente pode fazer duas requisições principais ao servidor, sendo cada uma delas uma conexão diferente. A primeira é enviando a mensagem "999!" caso queira carregar a lista de candidatos do servidor. A segunda ele envia a mensagem "888!" enviando os votos realizados em sua urna e logo após recebendo do servidor todos os votos que até então nele foram computados, incluído os que ele acabou de receber. Ainda nessa segunda opção, o servidor envia um 'ok!' para sinalizar que está preparado para receber os votos da urna (cliente), o cliente então envia os votos e recebe como resposta do servidor todos os votos computados até o momento. Veja a figura 1.

Além disso, é no cliente que toda a interface gráfica está codificada. Onde o usuário irá interagir com a urna eletrônica e irá realizar seu voto. O código se encontra disponível no github: https://github.com/dcardos/RedesBSI017

3.2 Código Java - Servidor

Neste arquivo, foi criado uma classe apenas com intuito de criar todos os candidatos, computar os votos de todas as urnas(clientes), criar as threads com intuito de ter o suporte de mais de um cliente acessar o servidor ao mesmo tempo. Além disso, o servidor também converte as strings para bytes a fim de enviar os dados pela conexão.

3.3 Código C - Servidor

Neste arquivo, assim como o servidor em java estabelecemos uma struct com 6 candidatos, sendo um deles sempre o 'candidato nulo'. Para a síntese desse servidor, utilizamos diversas biblioteca, são elas:

- ***sys/types.h*** *- Para definições de tipos*
- ***sys/socket.h*** *- Manipulação de sockets*
- ***netinet/in.h*** *- Define os padrões do protocolo IP*
- ***arpa/inet.h*** *- Converte endereços dos hosts*
- ***stdio.h*** *- operações de entrada/saída (printf, scanf, fprintf, fscanf)*
- ***stdlib.h*** *- alocação de memória, controle de processos, conversões etc*
- ***string.h*** *- manipulação de cadeias de caracteres*
- ***unistd.h*** *- access to the POSIX operating system API*
- ***errno.h*** *- identificar e relatar erros de execução*

E as principais funções utilizadas foram:

- ***socket****(sin_family,socket_type,protocol_number)*
- ***setsockopt****(int socket, int level, int optname, void*optval, size_t optlen)*
- ***bind****(int socket, struct sockaddr*addr, size_t length)*
- ***listen****(int socket, unsigned int n)*
- ***accept****(int socket, struct sockaddr*addr, size_t*length_ptr)*
- ***recv****(int socket, void *buffer, size_t size, int flags)*
- ***send****(int socket, void*buffer, size_t size, int flags)*

Foi possível perceber que o desenvolvimento em linguagem C é algo mais complexo se comparado com a linguagem java, uma vez que as funções e métodos

4. Metodologia, Ambientes e Ferramentas Utilizadas

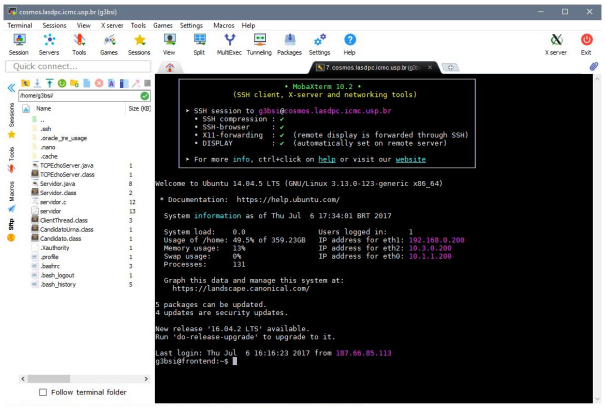

Figura 4. MobaXTream

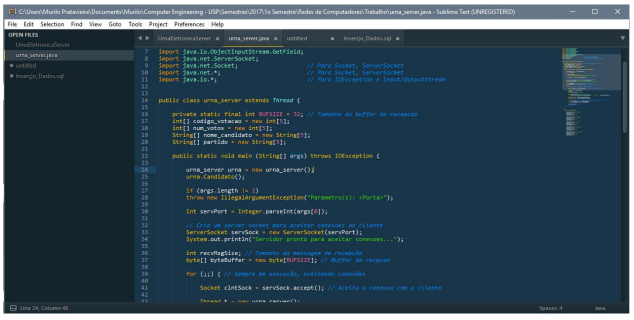

Figura 5. Sublime

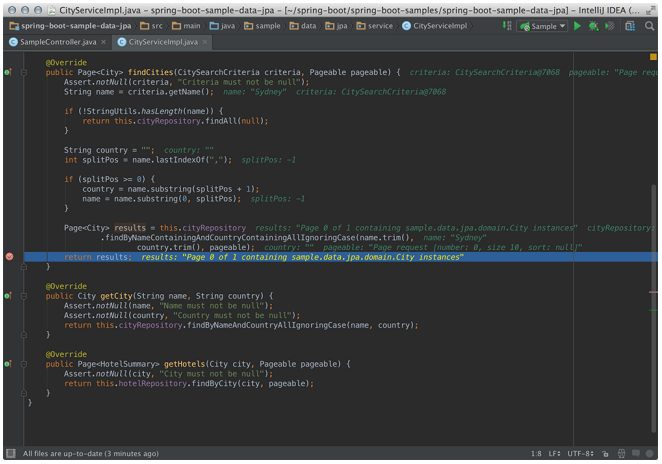

Figura 6. IntelliJ

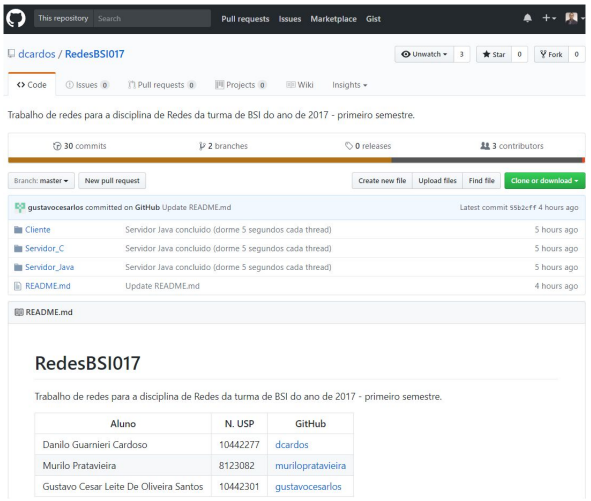

Figura 7. Git

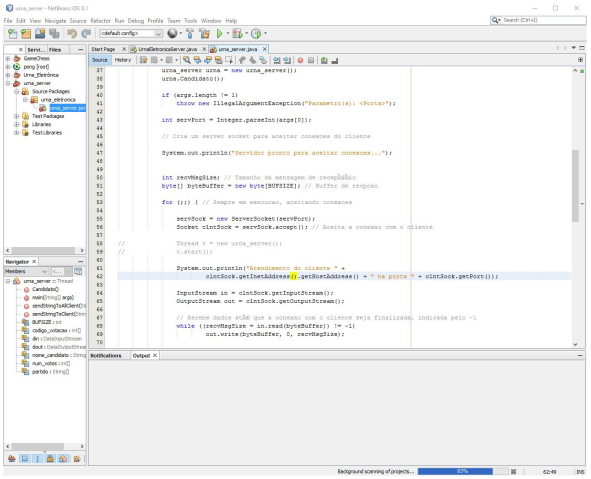

Figura 8. NetBeans

5. Modelagem do Sistema a Ser Desenvolvido

Protocolo e Interface Gráfico

Botões:

- **Carregar Candidatos**
- **Votar**
- **Votar em Branco**
- **Votar Nulo**
- **Enviar votos e Atualizar**
- **Encerrar**

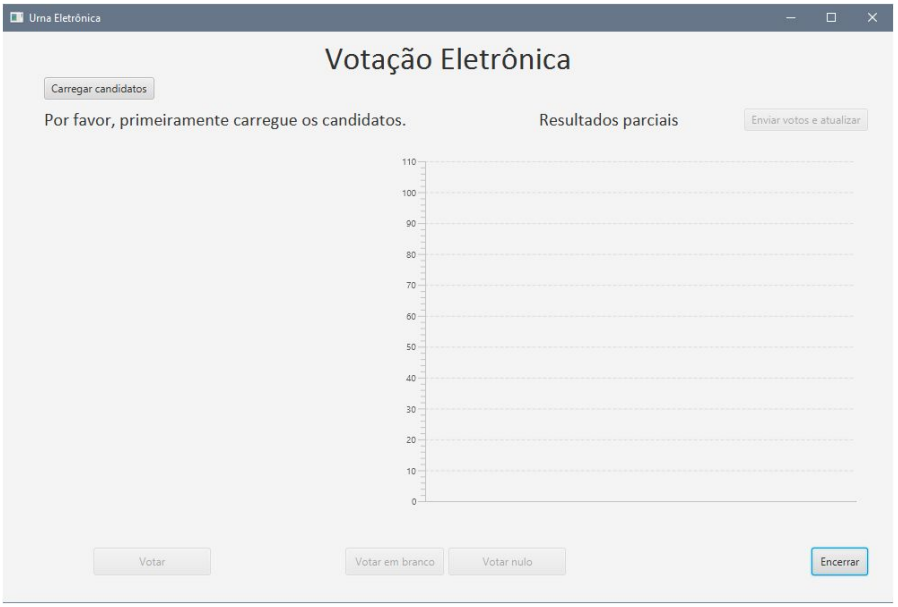

Figura 9. Início da Aplicação no Cliente

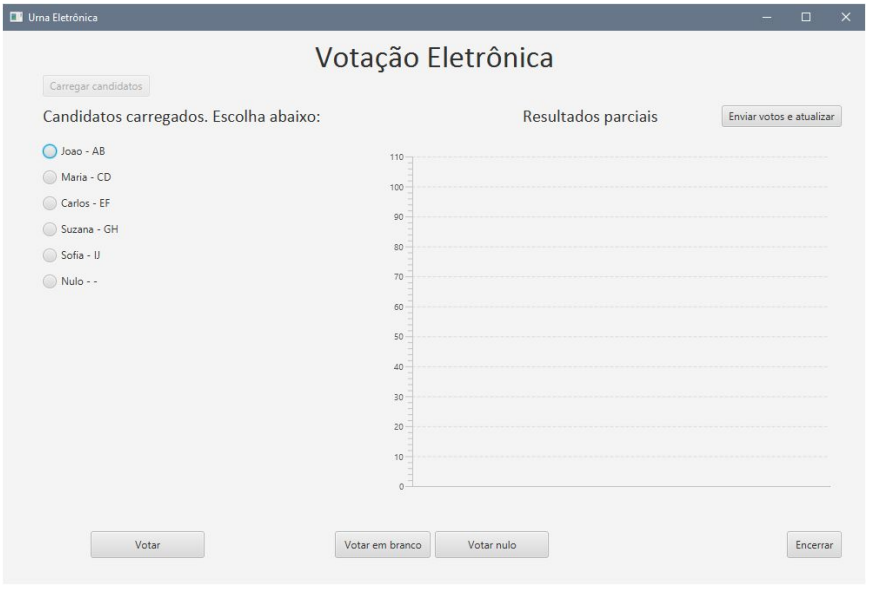

Figura 10. Candidatos Carregados no Servidor

Figura 11. Resultados da Votação no Cliente

6. Conclusão

Em conclusão, vale ressaltar que essa aplicação de urna eletrônica com socket abordou inúmeros conceitos sobre redes de computadores. Alguns assuntos como sockets, threads e protocolos ficaram mais evidenciados e exigiram mais aprofundamento para síntese desta aplicação da urna. Além disso, a interface gráfica, desde os botões e gráficos do cliente demonstrando também exigiram alguns testes para sincronização.

7. Referências

1. **COMPUTER NETWORKING: A TOP-DOWN APPROACH (7th Edition)**
2. **SLIDES DA AULA SSC-0142 REDES DE COMPUTADORES - PROF. JULIO CESAR ESTRELLA**
3. **HTTPS://STACKOVERFLOW.COM**
4. **WWW.TUTORIALSPOINT.COM/JAVA/JAVA_NETWORKING.HTM**
5. **HTTPS://CANALTECH.COM.BR/O-QUE-E/E-QUE-E/O-QUE-E-THREAD/**