

Sistema de Comunicação Simultânea de Usuários em Smart Grids

Execução: Individual

Data de entrega: 22 de Novembro de 2023 até 23h:59min

Introdução

Protocolo

Especificação das Mensagens

Fluxo das Mensagens de Controle

Descrição das Funcionalidades

Implementação

Comandos

Execução

Avaliação

Entrega

Prazo de Entrega

Dicas e Cuidados

Exemplos de execução

INTRODUÇÃO

Smart grid consiste em um sistema baseado em comunicação e tecnologia da informação adequado para a geração, fornecimento e consumo de energia. Esse sistema trata-se de redes inteligentes que são incorporadas às usinas a fim de recolher e administrar dados e, com base nestas informações coletadas, controlá-las com eficácia. As smart grids utilizam do fluxo bidirecional de informações, com intuito de formar um sistema automatizado, amplamente distribuído e disponível de novas funcionalidades. Estas aplicabilidades indicadas são: *controle, competência operacional, resiliência da rede e uma melhor integração de tecnologias renováveis*.

O Sistema de Supervisão e Controle (SCADA), ao ser incorporado a *smart grid*, possui o encargo de efetuar as coletas, supervisão e administração dos dados. Os dados obtidos geralmente referem-se a valores de medidas e status dos diversos componentes da rede, tornando o sistema uma parte fundamental do setor elétrico, mediante a sua capacidade de cobrir grandes áreas e executar comunicações em tempo real. O SCADA é amplamente empregado para supervisionar e monitorar continuamente infraestruturas críticas, como redes de distribuição de água, usinas de geração e distribuição de eletricidade, refinarias de petróleo, usinas nucleares e sistemas de transporte público.

Uma empresa de controle e automação do segmento de controle em *Smart Grids* necessita desenvolver um projeto piloto para prover um ambiente de controle de informações de rede. Este ambiente deve consultar o estado de diversas redes *smart grids* através de unidades de controle centralizadas, responsáveis por todo o fluxo de informações do sistema, as Unidades Terminais Principais, do inglês *Main Terminal Units* - MTUs (**os servidores**). Além disso, o ambiente deve dispor de Interfaces Humano-Máquina, do inglês *Human-Machine Interfaces* - HMIs (**os clientes**), cuja função é consultar dados dos dispositivos em campo, como sensores, e das redes *smart grids* por meio de protocolos de comunicação. Esta comunicação ocorre através da Internet. A Figura 1 ilustra a comunicação entre cada uma das entidades do projeto (MTUs e HMIs).

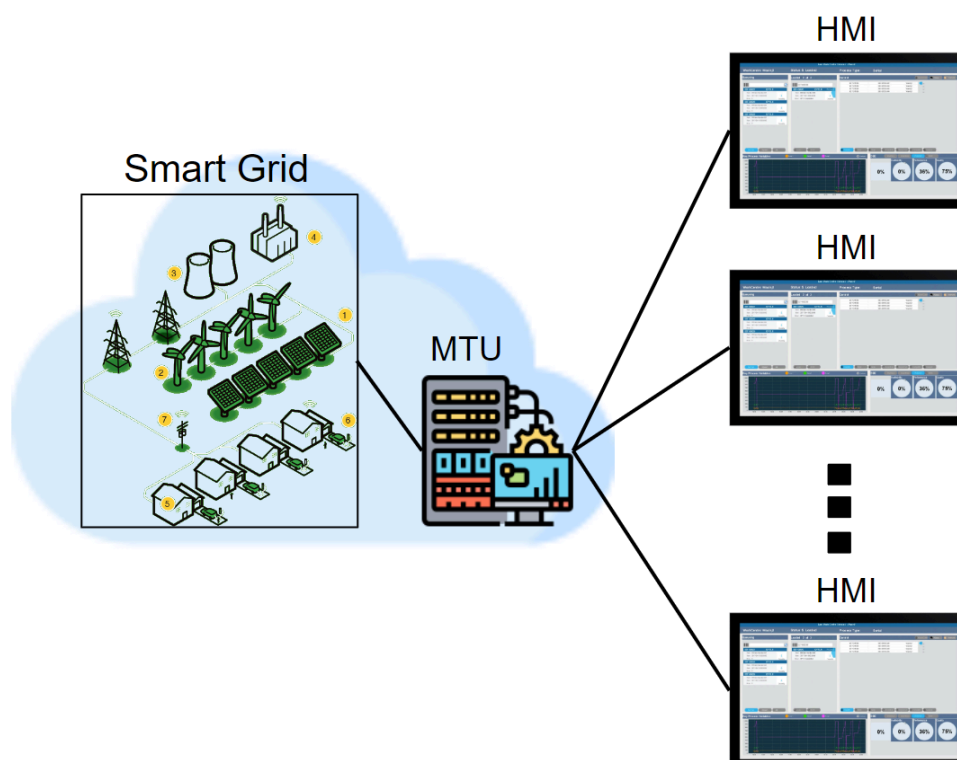


Figura 1 - Exemplo de comunicação entre as entidades do projeto

Nesse contexto, você foi contratado como Engenheiro de Automação para atuar no projeto em questão. A sua tarefa é desenvolver um sistema de comunicação entre os clientes HMIs que permite o monitoramento em tempo real de todas as redes smart grids. Esse sistema deve permitir a troca de informações e comandos entre os diferentes equipamentos e processos envolvidos, facilitando a coordenação dos recursos e otimizando o desempenho da linha de produção.

A topologia empregada, dada as restrições dos equipamentos utilizados, será de um servidor em cada rede smart grid, os quais conectam-se aos clientes de suas respectivas smart grids. Além disso, esses servidores se conectam para a troca de informação sobre o estado das smart grids que estão em seus respectivos controles, como observado na Figura 2.

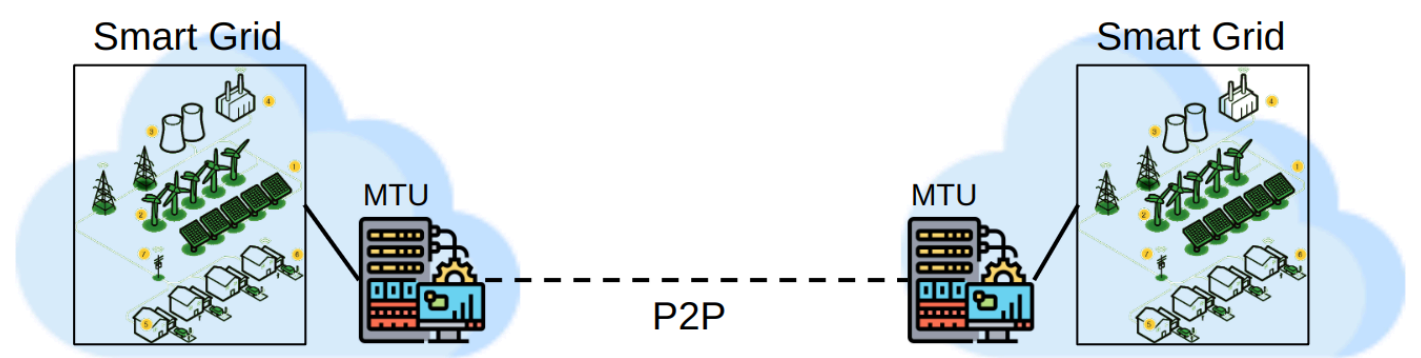


Figura 2 - Exemplo de comunicação entre os servidores

O programa servidor (MTU) deve armazenar dados **aleatórios** de potência e de eficiência energética de, pelo menos, 3 (três) e, no máximo, 10 (dez) sensores, de acordo com o exemplo na Tabela I.

Tabela I - Dados armazenados de sensores

Smart Grid 1		
ID	Potência = [0,1500]* W	Eficiência Energética = [0,100]* %
0	350 W	30%
1	227 W	100%
2	1359 W	67%

* Os valores aleatórios devem ser números inteiros que respeitem o intervalo estipulado para cada parâmetro.

Em outras palavras, você deve implementar um **sistema** caracterizado pela existência de dois **servidores (MTUs)** conectados entre si através de uma abordagem **peer-2-peer (P2P)**, i.e., onde os servidores podem realizar o papel **passivo ou ativo** ao se conectarem. Além disso, cada um desses servidores são responsáveis por estabelecer uma **conexão passiva** com os **clientes (HMIs)** em suas respectivas smart grids, pelo **gerenciamento de múltiplas conexões** com os seus clientes, pela **intermediação das mensagens** enviadas pelos clientes locais da smart grid que pertencem, pelo gerenciamento de **múltiplas conexões** com os seus clientes, pelo **encerramento passivo de conexão** com seus clientes e pelo **encerramento de conexão** com o seu peer.

Os **equipamentos HMIs** desempenham o papel de **clientes**, sendo responsáveis pelo **estabelecimento ativo de conexão** com os servidores de suas respectivas *smart grids*, pelo **envio e recebimento de mensagens** trocadas com o servidor e pelo **encerramento ativo de conexão**.

Toda conexão deve utilizar a interface de sockets na linguagem C.

Você desenvolverá os dois (2) programas para um sistema simples de troca de mensagens empregando apenas as funcionalidades da biblioteca de sockets POSIX e a comunicação via protocolo. O programa referente ao servidor deverá usar **dois** sockets, um socket para a conexão P2P com outro servidor e outro socket para a conexão passiva com os clientes. Deve-se utilizar a função **select()** da biblioteca de sockets para o gerenciamento das múltiplas conexões estabelecidas. As próximas seções detalham o que cada entidade (servidor e cliente) deve fazer.

Os objetivos gerais deste trabalho são:

1. Implementar servidor usando a interface de sockets na linguagem C;
2. Implementar cliente usando a interface de sockets na linguagem C;
3. Escrever o relatório.

PROTOCOLO

O protocolo de aplicação deve funcionar sobre o protocolo TCP. Isso implica que as mensagens são entregues sobre um canal de bytes com garantias de entrega em ordem, mas é sua responsabilidade implementar as especificações das mensagens e as funcionalidades tanto dos servidores quanto dos clientes.

Os servidores e os clientes trocam mensagens curtas de até 500 bytes usando o transporte TCP. As mensagens carregam textos codificados segundo a tabela ASCII. Apenas letras, números e espaços podem ser transmitidos. Caracteres acentuados e especiais não devem ser transmitidos.

Especificações das Mensagens

Esta seção especifica as mensagens padrões na comunicação de controle e dados da rede, bem como as mensagens de erro e confirmação. Nas tabelas abaixo, as células em “–” correspondem aos campos que não precisam ser definidos nas mensagens.

Mensagens de Controle		
Tipo	Payload	Descrição
REQ_ADDPEER	–	Mensagem Peer-2-Peer de requisição de conexão entre peers
REQ_DCPEER	PidM _i	Mensagem Peer-2-Peer de requisição para o encerramento de conexão entre peers
RES_ADDPEER	PidM _i	Mensagem de resposta de conexão entre peers com identificação PidM _i do servidor M _i
REQ_ADD	–	Mensagem de requisição de entrada de cliente na rede

REQ_DC	IdC _i	Mensagem de requisição de saída de cliente na rede, onde IdC _i corresponde a identificação do cliente solicitante.
RES_ADD	IdC _i	Mensagem de resposta de identificação IdC _i do cliente C _i

Mensagens de Dados		
Tipo	Payload	Descrição
REQ_LS	–	Mensagem de requisição de sensor com maior potência útil na rede local
REQ_ES	–	Mensagem de requisição de sensor com maior potência útil na rede externa
REQ_LP	–	Mensagem de requisição de potência útil da rede local
REQ_EP	–	Mensagem de requisição de potência útil da rede externa
REQ_MS	–	Mensagem de requisição de sensor com maior potência útil em ambas as redes
REQ_MN	–	Mensagem de requisição de rede com maior potência útil
RES_LS	PidM_isensorId_i; pot_util _i (pot _i efic_energ _i)	Mensagem de resposta de sensor com maior potência útil na rede local
RES_ES	PidM_jsensorId_j; pot_util _j (pot _j efic_energ _j)	Mensagem de resposta de sensor com maior potência útil na rede externa
RES_LP	PidM_ipot_util_i	Mensagem de resposta de potência útil na rede local
RES_EP	PidM_jpot_util_j	Mensagem de resposta de potência útil na rede externa
RES_MS	PidM_isensorId_i; pot_util _i (pot _i efic_energ _i)	Mensagem de resposta de sensor com maior potência útil em ambas as redes
RES_MN	PidM_ipot_util_i	Mensagem de resposta de rede com maior potência útil

Mensagens de Erro ou Confirmação

Tipo	Payload	Descrição
ERROR	Code	Mensagem de erro transmitida do Servidor para cliente C_i . O campo payload informa o código de erro. Abaixo descrição de cada código: 01 : " <i>Client limit exceeded</i> " 02: " <i>Peer limit exceeded</i> " 03: " <i>Peer not found</i> ". 04: " <i>Client not found</i> ".
OK	Code	Mensagem de confirmação transmitida do Servidor para cliente C_i . O campo payload informa o código de confirmação. Abaixo descrição de cada código: 01 : " <i>Successful disconnect</i> ".

Fluxo das Mensagens de Controle

Esta seção descreve o fluxo de mensagens de controle transmitidas entre servidor-servidor e entre cliente-servidor a fim de coordenar a comunicação dos clientes na rede. Além das decisões e impressões em tela realizadas pelos servidores e clientes.

Abertura de comunicação entre Servidores (Peer-2-Peer)

1. O servidor M_i tenta solicitar ao servidor M_j a abertura de comunicação por meio da mensagem **REQ_ADDPEER**
 - 1.1. Caso não haja um servidor M_j aberto à conexão, o servidor M_i imprime em tela a mensagem "*No peer found, starting to listen..*" e começa a ouvir possíveis novas conexões.
 - 1.2. Caso haja um servidor M_j aberto à conexão, o mesmo recebe a requisição de M_i e verifica se a quantidade máxima de conexões peer-2-peer foi alcançada.
 - 1.2.1. Em caso positivo, o servidor M_j responde uma mensagem de **ERROR(02)** para M_i
 - 1.2.1.1. O servidor M_i recebe a mensagem **ERROR(02)** e imprime na tela a descrição do código de erro correspondente (vide *Especificação das Mensagens*)
 - 1.2.2. Em caso negativo, o servidor M_j define um identificador **PidM_i** para M_i , registra o identificador em sua base de dados, imprime em tela a mensagem "*Peer PidM_i connected*" e envia para M_i o identificador **PidM_i** definido através da mensagem **RES_ADDPEER(PidM_i)**.
 - 1.2.2.1. O servidor M_i recebe o seu identificador através da mensagem **RES_ADDPEER(PidM_i)**, imprime na tela "*New Peer ID: PidM_i*" define um identificador **PidM_j** para o servidor M_j , imprime em tela a mensagem "*Peer PidM_j connected*" e envia para M_j o identificador através da mensagem **RES_ADDPEER(PidM_j)**
 - 1.2.2.2. O servidor M_j recebe o seu identificador através da mensagem **RES_ADDPEER(PidM_j)**, imprime na tela "*New Peer ID: PidM_j*".

Fechamento de comunicação entre Servidores (Peer-2-Peer)

1. Um servidor M_i recebe comando via teclado **kill** e solicita ao servidor M_j o fechamento de comunicação por meio da mensagem **REQ_DCPEER(PidM_j)**.
2. Servidor M_j recebe **REQ_DCPEER(PidM_j)** e verifica se **PidM_j** é o identificador do peer conectado.
 - 2.1. Em caso negativo, o servidor M_j responde mensagem de erro **ERROR(03)** para o servidor M_i .
 - 2.1.1. Servidor M_i recebe mensagem de **ERROR(03)** de M_j e imprime na tela a descrição do código de erro correspondente (vide *Especificação das Mensagens*).
 - 2.2. Em caso positivo, o servidor M_j remove M_i da sua base de dados, responde mensagem **OK(01)** para M_i , imprime na tela a mensagem *Peer PidM_j disconnected*. Servidor M_i recebe mensagem **OK(01)** de confirmação, imprime em tela a descrição da mensagem, remove M_j da sua base de dados, imprime na tela a mensagem *Peer PidM_j disconnected*.

Abertura de comunicação de Cliente com Servidor

1. Um cliente C_i solicita ao servidor M_i a abertura de comunicação a fim de obter seu identificador na rede.
2. O servidor M_i recebe requisição de C_i e verifica se quantidade máxima de conexões foi alcançada.
 - 2.1. Em caso positivo, o servidor M_i **imprime a** mensagem de erro **ERROR(01)** (vide *Especificação das Mensagens*).
 - 2.1.1. Cliente C_i **imprime** em tela descrição do código de erro **ERROR(01)**.
 - 2.2. Em caso negativo, **cliente C_i envia a mensagem REQ_ADD para o servidor M_i** .
 - 2.2.1. O servidor M_i define um identificador **IdC_i** para C_i único entre os seus clientes, registra o identificador em sua base de dados, imprime em tela a mensagem *"Client IdC_i added"* e envia para o cliente C_i a mensagem **RES_ADD(IdC_i)**. O cliente C_i , ao receber mensagem **RES_ADD(IdC_i)** do **servidor M_i** , registra sua nova identificação e imprime em tela a mensagem *New ID: IdC_i*.

Fechamento de comunicação de Cliente com Servidor

1. Um cliente C_i (HMI) recebe comando via teclado **kill** e solicita ao servidor M_i o fechamento da comunicação por meio da mensagem **REQ_DC(IdC_i)**.
2. O servidor M_i recebe **REQ_DC(IdC_i)** e verifica se **IdC_i** existe na base de dados.
 - 2.1. Em caso negativo, o servidor M_i responde mensagem de erro **ERROR(04)** para cliente C_i .
 - 2.1.1. Cliente C_i recebe mensagem **ERROR(04)** de servidor M_i e imprime em tela a descrição do código de erro correspondente (vide *Especificação das Mensagens*).
 - 2.2. Em caso positivo, o servidor M_i remove C_i da base de dados, responde mensagem **OK(01)** para C_i , desconecta C_i , imprime em tela a mensagem *Client IdC_i removed*. Cliente C_i recebe mensagem **OK(01)** de confirmação e imprime em tela a descrição da mensagem, fecha a conexão e encerra a execução.

Descrição das Funcionalidades

Esta seção descreve o fluxo de mensagens transmitidas entre os clientes (HMI) e os servidores (MTU) resultante de cada uma das seis funcionalidades da aplicação a fim de monitorar os sensores nas redes elétricas.

1) Consultar sensor local com maior potência útil

1. Um cliente **C_i** (HMI) recebe comando via teclado
`show localmaxsensor`
para consultar os valores do sensor com maior potência útil na rede local. Para isso, o cliente **C_i** envia a mensagem **REQ_LS** para o servidor **M_i** (MTU).
2. O servidor **M_i** recebe a solicitação, consulta a **Tabela de Sensores** de sua rede local e calcula a potência útil ($\text{potência} \times \text{eficiência_energética} / 100$) de cada sensor instalado.
 - 2.1. O servidor **M_i** responde o cliente **C_i** com os valores do sensor com maior potência útil por meio da mensagem **RES_LS**.
 - 2.1.1. O cliente **C_i** recebe mensagem e imprime em tela:
`local PidMi sensor sensorIdi: pot_util, (pot, efic_energi)`

2) Consultar sensor externo com maior potência útil

1. Um cliente **C_i** recebe comando via teclado
`show externalmaxsensor`
para consultar os valores do sensor com maior potência útil na rede externa. Para isso, o cliente **C_i** envia a mensagem **REQ_ES** para o servidor **M_i**.
2. O servidor **M_i** recebe a solicitação e envia a mensagem **REQ_ES** para o servidor externo **M_j**.
 - 2.1. O servidor **M_j** recebe a solicitação, consulta a **Tabela de Sensores** de sua rede local e calcula a potência útil ($\text{potência} \times \text{eficiência_energética} / 100$) de cada sensor instalado.
 - 2.1.1. O servidor **M_j** responde o servidor **M_i** com os valores do sensor com maior potência útil por meio da mensagem **RES_ES**.
 - 2.2. O servidor **M_i** recebe a resposta do servidor **M_j** e envia a mensagem **RES_ES** para o cliente **C_i**.
 - 2.2.1. O cliente **C_i** recebe a mensagem e imprime em tela:
`external PidMj sensor sensorIdi: pot_util, (pot, efic_energi)`

3) Consultar potência útil da rede local

1. Um cliente **C_i** recebe comando via teclado
`show localpotency`
para consultar a potência útil de sensores instalados na rede local. Para isso, o cliente **C_i** envia a mensagem **REQ_LP** para o servidor **M_i**.
2. O servidor **M_i** recebe a solicitação, consulta a **Tabela de Sensores** de sua rede local e calcula o somatório da potência útil ($\text{potência}_a \times \text{eficiência_energética}_a / 100 + \text{potência}_b \times \text{eficiência_energética}_b / 100 + \dots + \text{potência}_n \times \text{eficiência_energética}_n / 100$) de todos os sensores instalados na rede.
 - 2.1. O servidor **M_i** responde o cliente **C_i** com o valor do somatório das potências úteis de sensores por meio da mensagem **RES_LP**.
 - 2.1.1. O cliente **C_i** recebe mensagem e imprime em tela:
`local PidMi potency: pot_util`

4) Consultar potência útil da rede externa

1. Um cliente C_i recebe comando via teclado
`show externalpotency`
para consultar a potência útil de sensores instalados na rede externa. Para isso, o cliente C_i envia a mensagem **REQ_EP** para o servidor M_i .
2. O servidor M_i recebe a solicitação e envia a mensagem **REQ_EP** para o servidor externo M_j .
 - 2.1. O servidor M_j recebe a solicitação, consulta a **Tabela de Sensores** de sua rede local e calcula o somatório da potência útil ($\text{potência}_a * \text{eficiência_energética}_a / 100 + \text{potência}_b * \text{eficiência_energética}_b / 100 + \dots + \text{potência}_n * \text{eficiência_energética}_n / 100$) de todos os sensores instalados em sua rede.
 - 2.1.1. O servidor M_j responde o servidor M_i com o valor do somatório das potências úteis de sensores por meio da mensagem **RES_EP**.
 - 2.2. O servidor M_i recebe a resposta do servidor M_j e envia a mensagem **RES_EP** para o cliente C_i .
 - 2.2.1. O cliente C_i recebe a mensagem e imprime em tela:
`external PidMj potency: pot_utilj`

5) Consultar sensor com maior potência útil em ambas as redes

1. Um cliente C_i recebe comando via teclado
`show globalmaxsensor`
para consultar os valores do sensor com maior potência útil em ambas as redes. Para isso, o cliente C_i envia a mensagem **REQ_MS** para o servidor M_i .
2. O servidor M_i recebe a solicitação, consulta a **Tabela de Sensores** de sua rede local e calcula a potência útil ($\text{potência} * \text{eficiência_energética} / 100$) de cada sensor instalado e armazena os valores do sensor com maior potência útil na rede local.
3. O servidor M_i envia a mensagem **REQ_ES** para o servidor externo M_j .
 - 3.1. O servidor M_j recebe a solicitação, consulta a **Tabela de Sensores** de sua rede local e calcula a potência útil ($\text{potência} * \text{eficiência_energética} / 100$) de cada sensor instalado.
 - 3.1.1. O servidor M_j responde o servidor M_i com os valores do sensor com maior potência útil por meio da mensagem **RES_ES**.
 - 3.2. O servidor M_i recebe a resposta do servidor M_j e compara as potências úteis dos sensores local e externo com maior potência útil em suas respectivas redes.
 - 3.3. O servidor M_i responde o cliente com os valores do sensor com maior potência útil em ambas as redes por meio da mensagem **RES_MS**.
 - 3.3.1. O cliente C_i recebe mensagem e imprime em tela:
`global PidMi sensor sensorIdi: pot_utili (poti efic_energi)`

6) Consultar rede com maior potência útil

1. Um cliente C_i recebe comando via teclado
`show globalmaxnetwork`
para consultar a rede com maior potência útil de sensores instalados. Para isso, o cliente C_i envia a mensagem **REQ_MN** para o servidor M_i .

2. O servidor **M_i** recebe a solicitação, consulta a **Tabela de Sensores** de sua rede local e calcula o somatório da potência útil ($\text{potência}_a * \text{eficiência_energética}_a / 100 + \text{potência}_b * \text{eficiência_energética}_b / 100 + \dots + \text{potência}_n * \text{eficiência_energética}_n / 100$) de todos os sensores instalados na rede local.
3. O servidor **M_i** envia a mensagem **REQ_EP** para o servidor externo **M_j**.
 - 3.1. O servidor **M_j** recebe a solicitação, consulta a **Tabela de Sensores** de sua rede local e calcula o somatório da potência útil ($\text{potência}_a * \text{eficiência_energética}_a / 100 + \text{potência}_b * \text{eficiência_energética}_b / 100 + \dots + \text{potência}_n * \text{eficiência_energética}_n / 100$) de todos os sensores instalados na rede.
 - 3.1.1. O servidor **M_j** responde o servidor **M_i** com o valor do somatório das potências úteis de sensores por meio da mensagem **RES_EP**.
 - 3.2. O servidor **M_i** recebe a resposta do servidor **M_j** e compara o somatório das potências úteis das redes local e externa.
 - 3.3. O servidor **M_i** responde o cliente **C_i** com o valor do somatório das potências úteis de sensores da rede com maior potência útil por meio da mensagem **RES_MN**.
 - 3.3.1. O cliente **C_i** recebe mensagem e imprime em tela:
`global PidMi potency: pot_util;`

IMPLEMENTAÇÃO

Os seguintes detalhes devem ser observados no desenvolvimento de cada programa que fará parte do sistema. É importante observar que o protocolo é simples e único (o cliente sempre tem que enviar a mensagem para o servidor e vice-versa ou o servidor tem que enviar a mensagem para outro servidor, de modo que o correto entendimento da mensagem deve ser feito por todos os programas).

Como mencionado anteriormente, a implementação do protocolo da aplicação utilizará a comunicação TCP. Haverá um socket em cada **cliente**, independente de quantos outros programas se comunicarem com aquele processo. Já os servidores inicializam-se com **dois** sockets, um para a conexão com o servidor peer e uma para a conexão com os clientes. À medida que se conecta e se desconecta de clientes, outros sockets são adicionados/descartados do seu pool de sockets.

O tipo de endereço IP assumido neste trabalho prático deve ser IPv4. Um número máximo de **2 servidores serão executados simultaneamente**. Cada **servidor** deve tratar até **10 clientes simultaneamente**. Ademais, o servidor é responsável por definir identificações únicas para cada cliente na rede. Um **cliente** inicia sem identificação, após a solicitação de entrada na rede ele recebe sua identificação. Além disso, o cliente e o servidor devem receber mensagens do teclado.

Outros detalhes de implementação:

- O servidor deve encerrar apenas a conexão com o cliente ao receber a mensagem **“kill”** a qualquer momento
- Cada mensagem possui no máximo 500 bytes

Execução

Seu servidor deve receber **um endereço IPv4 para tentar se conectar ativamente (tcp) ao peer** e dois números de porta na linha de comando especificando em qual porta ele vai estabelecer a conexão peer-2-peer e em qual vai receber conexões dos clientes. Para padronização do trabalho, utilize a porta 90900 para a conexão peer-2-peer e as portas **90100** e **90200** para se conectar com clientes. Seu

cliente deve receber, **estritamente nessa ordem**, o endereço IP e a porta do servidor em que ele deseja se conectar para o estabelecimento da comunicação. Para realizar múltiplas conexões de clientes com o servidor basta executar múltiplas vezes o código do programa cliente.

A seguir, um exemplo de execução de dois clientes conectados com um servidor em quatro terminais distintos:

Terminal 1: `./server 127.0.0.1 90900 90100`

Terminal 2: `./server 127.0.0.1 90900 90200`

Terminal 3: `./client 127.0.0.1 90100`

Terminal 4: `./client 127.0.0.1 90200`

AVALIAÇÃO

O trabalho deve ser realizado individualmente e **deve ser implementado na linguagem de programação C** utilizando somente a biblioteca padrão (interface POSIX de sockets de redes). Deve ser possível executar seu programa no sistema operacional **Linux** e **não deve utilizar bibliotecas Windows, como o winsock**. Procure escrever seu código de maneira clara, com comentários pontuais e bem indentados. Isto facilita a correção dos monitores e tem impacto positivo na avaliação.

Correção Semi-automática

Seu servidor será corrigido de forma semi-automática por uma bateria de testes. Cada teste verifica uma funcionalidade específica do servidor. Os testes avaliam a aderência do seu servidor ao protocolo de comunicação inteiramente através dos dados trocados através da rede (a saída dos seus programas na tela, e.g., para depuração, não impacta os resultados dos testes).

Para a correção os seguintes testes serão realizados (**com IPv4**):

- Abertura de comunicação cliente-servidor: **+2 pontos**
- Fechamento de comunicação cliente-servidor: **+2 pontos**
- Abertura de comunicação servidor-servidor: **+3 pontos**
- Fechamento de comunicação servidor-servidor: **+3 pontos**
- Consulta sensor local com maior potência útil: **+1 ponto**
- Consulta sensor externo com maior potência útil: **+2 pontos**
- Consulta potência útil da rede local: **+1 ponto**
- Consulta potência útil da rede externa: **+2 pontos**
- Consulta sensor com maior potência útil em ambas as redes: **+3 pontos**
- Consulta rede com maior potência útil: **+3 pontos**
- Cliente envia kill para o servidor e encerra a execução do programa cliente: **+2 pontos**

Total: **24 pontos (80%)** + 6 pontos (20%) (documentação)

Entrega

Cada aluno deve entregar documentação em PDF de até 6 páginas, sem capa, utilizando fonte tamanho 10, e figuras de tamanho adequado ao tamanho da fonte. **Ele deve conter uma descrição da arquitetura adotada para o servidor, os refinamentos das ações identificadas no mesmo, as estruturas de dados utilizadas, as decisões de implementação não documentadas nesta especificação.** Como sugestão, considere incluir as seguintes seções no relatório: introdução, mensagens, arquitetura, servidor, cliente,

discussão e conclusão. O relatório deve ser entregue em formato PDF. A documentação corresponde a 20% dos pontos do trabalho (**+6 pontos**), mas só será considerada para as funcionalidades implementadas corretamente.

Será utilizado um sistema para detecção de código repetido, portanto não é admitido cola de trabalhos. Se uma das partes do trabalho não for entregue (código ou relatório) a nota final será zero.

Cada aluno deve entregar, além da documentação, o **código fonte em C** e um **Makefile** para compilação do programa. Instruções para submissão:

- O Makefile deve compilar o “client” e o “server”.
- Seu código deve ser compilado pelo comando “make” sem a necessidade de parâmetros adicionais.
- A entrega deve ser feita no formato ZIP, seguindo a nomenclatura: TP_MATRICULA.zip
- O nome dos arquivos deve ser padronizado:
 - server.c
 - client.c
 - common.c, common.h (se houver)

Prazo de entrega

Os trabalhos poderão ser entregues até às 23:59 (vinte e três e cinquenta e nove) do dia especificado para a entrega. O horário de entrega deve respeitar o relógio do sistema Moodle, ou seja, a partir de 00:00 do dia seguinte à entrega no relógio do Moodle, os trabalhos **não poderão ser entregues. Logo não serão considerados trabalhos entregues fora do prazo definido.**

Pedidos de Revisão de Notas

Os alunos poderão realizar pedidos de revisão de nota em até três dias corridos contados a partir da data da liberação da nota. Em caso de solicitação de revisão, o aluno deverá enviar o pedido para o e-mail do professor com as seguintes orientações:

- No título do email o rótulo [TP2-Rev] - nome e número de matrícula.
- No corpo da mensagem a sua questão propriamente dita.

Dicas e Cuidados

- O guia de programação em rede do Beej (<http://beej.us/guide/bgnet/>) tem bons exemplos de como organizar um servidor
- Procure escrever seu código de maneira clara, com comentários pontuais e bem indentado.
- Não se esqueça de conferir se seu código não possui erros de compilação ou de execução.
- Implemente o trabalho por partes. Por exemplo, implemente o tratamento das múltiplas conexões, depois crie os formatos das mensagens e, por fim, trate as mensagens no servidor ou no cliente.

Exemplos de execução

Exemplos de execução serão disponibilizados brevemente.