

Sistema de Monitoramento de Falhas em Redes Elétricas (v1.2)

Introdução

Motivados pelos recentes acontecimentos em Portugal e Espanha em que diversas cidades enfrentaram interrupções totais no fornecimento de energia elétrica. Levando em conta que tais acontecimentos poderão ocorrer no Brasil, os estudantes da disciplina de Redes de Computadores da UFMG, como uma forma de consolidar os conhecimentos teóricos e práticos, receberam uma missão desafiadora em sua trajetória acadêmica: projetar e testar uma solução capaz de prever possíveis panes e apagões em redes elétricas em regiões que carecem de monitoramento adequado.

Com o advento dos microchips e a popularização de tecnologias de baixo custo, as redes de sensores sem fio (WSNs) tornaram-se uma alternativa viável e democrática para diversas aplicações que exigem sensoramento e ações, como por exemplo as infra-estruturas críticas. Essas redes, associadas ao modelo TCP/IP, dão origem a Internet de Coisas (do inglês, Internet of Things - IoT), sendo essenciais para as indústrias 4.0 e 5.0. Elas são compostas por pequenos dispositivos embarcados com sensores que, apesar de suas limitações computacionais, conseguem coletar e transmitir informações do ambiente em tempo real. Atualmente, WSNs são amplamente empregadas em diferentes ecossistemas, desde o monitoramento de plantações agrícolas até a automação de fábricas.

Uma aplicação de grande impacto, considerando as infra-estruturas críticas, é justamente o monitoramento de redes elétricas, dando origem a conceitos como o Smart Grids. Em muitas regiões do mundo, as falhas e as quedas de energia têm causado transtornos consideráveis — afetando serviços essenciais, causando prejuízos econômicos e, em alguns casos, colocando vidas em risco. Situações recentes ilustram essa vulnerabilidade: tempestades e sobrecargas provocaram blecautes em grandes centros urbanos; falhas em linhas de transmissão resultaram em apagões em estados inteiros; e eventos climáticos extremos vêm testando a resiliência das infraestruturas elétricas.

Objetivo

Diante desse cenário, o objetivo deste trabalho é desenvolver e testar uma arquitetura de comunicação capaz de mitigar os danos causados por panes e apagões inesperados. A proposta envolve a utilização de dois servidores que se comunicarão entre si:

- Um servidor será responsável por armazenar e gerenciar informações sobre a localização dos sensores espalhados pela rede elétrica;

- O outro servidor será responsável por armazenar e atualizar o status da rede elétrica, monitorando indicadores de anomalias, como flutuações de tensão, sobrecargas, aquecimento de transformadores e interrupções de corrente detectadas pelos sensores.

Arquitetura do Sistema

A rede é composta por sensores distribuídos em pontos estratégicos da rede elétrica, capazes de detectar anomalias como:

- Variações anormais de tensão;
- Picos de corrente;
- Aquecimento excessivo de transformadores;
- Oscilações fora da faixa esperada de frequência.

Contudo, a análise bruta desses dados e sua conversão em diagnósticos binários (risco ou não) será feita por uma equipe externa. Para este projeto, assume-se que os servidores recebem apenas a informação final, ou seja, um status binário:

- **1** = risco detectado
- **0** = sem risco detectado

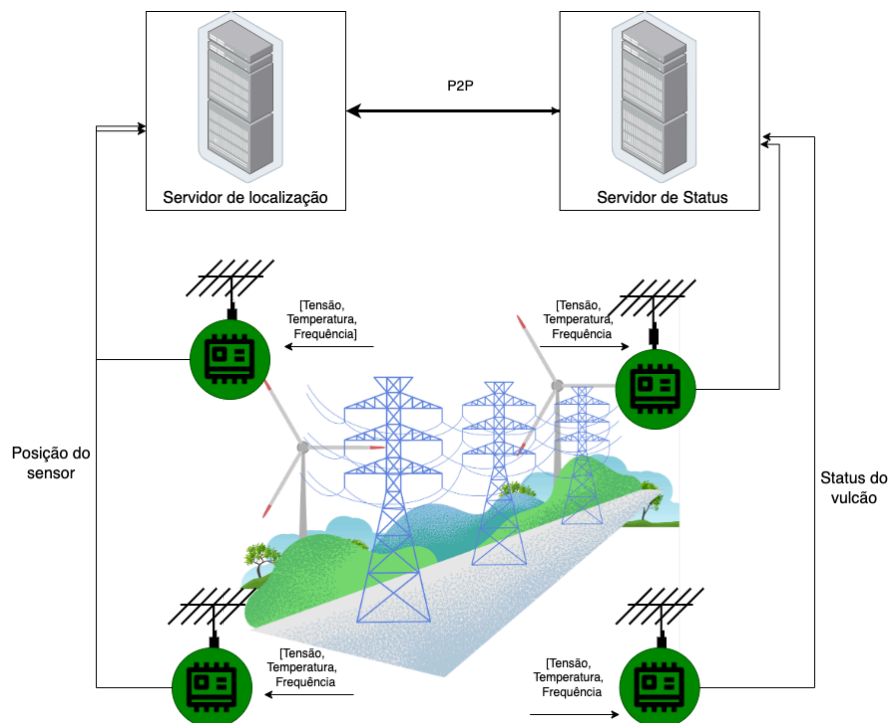


Figura 1 – Exemplo de cenário da arquitetura de monitoramento de falhas em uma rede elétrica, ilustrando a coleta de informações por sensores e servidores que integram a infraestrutura.

O programa **servidor de Status (SS)** deve armazenar os IDs dos sensores cadastrados, e incluir uma flag informando se aquele sensor identificou alguma anomalia, conforme o exemplo na **Tabela I**. Já o programa **servidor de localização (SL)** deve guardar a última localização das pessoas, conforme o exemplo na **Tabela II**.

Tabela I - Dados armazenados no servidor de Status

Exemplos	ID do usuário (10 caracteres numéricos)	Risco detectado (0 ou 1)
1	2042010377	0
2	2042980421	1
3	2041567983	0

Tabela II - Dados armazenados no servidor de localização

Exemplos	ID do usuário (10 caracteres numéricos)	localização (int) ✖
1	2042010377	1
2	2042980421	2
3	2041567983	-1

✖ O valor da localização deve ser um inteiro **entre 1 e 10** para indicando os pontos de instalação dos sensores ao redor do vulcão.

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 utilizando apenas as funcionalidades da biblioteca de sockets POSIX e a comunicação via protocolo. O programa referente ao servidor deverá usar **dois** sockets, um para a conexão com o outro servidor e outro 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 ou cliente) deve fazer.

Os objetivos gerais deste trabalho são:

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

Cluster Elétrico

O seu dever é identificar quais regiões da rede elétrica podem estar com algum tipo de pane elétrica. Portanto os códigos de localização são indicativos de quais áreas um dado nó está monitorando.

As regiões que compõem a rede elétrica monitorada são divididas em:

- **Área Norte:** localização 1, 2 e 3
- **Área Sul:** localização 4 e 5
- **Área Leste:** localização 6 e 7
- **Área Oeste:** localização 8, 9 e 10

Essa classificação permite identificar a posição aproximada de cada sensor com base apenas no valor numérico da sua última localização. Com isso, o sistema pode aplicar estratégias específicas de monitoramento e resposta conforme a área coberta. Sensores cuja última localização registrada seja igual a -1 são considerados fora da zona ativa de cobertura, e portanto, sem área definida.

Uma pane elétrica é detectada e alertada quando mais de 2 sensores naquela área são identificados com status de risco detectado.

Sua tarefa será projetar, implementar e avaliar essa arquitetura de comunicação, simulando um ambiente de monitoramento de redes elétricas utilizando redes de sensores sem fio. Esse sistema deve permitir a troca de informações e de comandos entre os diferentes equipamentos e processos envolvidos.

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 e (Direção)	Código	Payload	Descrição
REQ_CONNPEER (S _i -> S _j)	20	–	Mensagem de requisição de conexão entre os peers
RES_CONNPEER (S _j -> S _i)	21	PidS _i	Mensagem de resposta de conexão entre peers
REQ_DISCPEER (S _i -> S _j)	22	PidS _i	Mensagem de requisição de encerramento de conexão entre peers
REQ_CONNSEN (C _i -> S _i)	23	IDSen	Mensagem de requisição de entrada de sensor na rede
RES_CONNSEN (S _i -> C _i)	24	IdSen _i	Mensagem de resposta de entrada de sensor na rede, com a identificação IdC _i do sensor Sen _i
REQ_DISCSEN (C _i -> S _i)	25	IdSen _i	Mensagem de requisição de saída de sensor na rede, onde IdS _i corresponde à identificação do sensor solicitante

Mensagens de Dados			
Tipo e (Direção)	Código	Payload	Descrição
REQ_CHECKALERT (SS-> SL)	36	SensID	Mensagem de requisição de alerta de status positivo de pane na rede contendo o ID do sensor
RES_CHECKALERT (SL-> SS)	37	LocID	Mensagem de resposta de alerta de status positivo de pane na rede contendo a localização do sensor.
REQ_SENSLOC (C _i -> SL)	38	SensID	Mensagem de requisição de informação sobre a localização de um sensor
RES_SENSLOC (SL -> C _i)	39	LocId	Mensagem de resposta de informação sobre a localização de um sensor
REQ_SENSSTATUS (C _i -> SS)	40	SensID	Mensagem de requisição de status de sensor
RES_SENSSTATUS (C _i -> SS)	41	StatusID	Mensagem de resposta de status de sensor
REQ_LOCLIST (C _i -> SL)	42	LocId	Mensagem de requisição de sensores presentes em uma localização
RES_LOCLIST (SL -> C _i)	43	SenIDs	Mensagem de resposta de sensores presente em uma localização

Mensagens de Erro ou Confirmação			
Tipo	Código	Payload	Descrição
ERROR	255	Code	<p>Mensagem de erro transmitida do Servidor para sensor Sen_i. O campo payload informa o código de erro. Abaixo descrição de cada código:</p> <p>01: "Peer limit exceeded"</p> <p>02: "Peer not found"</p> <p>09: "Sensor limit exceeded"</p> <p>10: "Sensor not found"</p> <p>11: "Location not found"</p>

OK	0	Code	<p>Mensagem de confirmação transmitida do Servidor para sensor Sen_i. O campo payload informa o código de confirmação. Abaixo descrição de cada código:</p> <p>01: "Successful disconnect"</p> <p>02: "Successful create"</p> <p>03: "Status do sensor 0"</p>

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 S_i tenta se conectar ao servidor S_j enviando a mensagem **REQ_CONNPEER()**
 - 1.1. Caso não haja um servidor S_j aberto para conexão, o servidor S_i imprime em tela a mensagem *"No peer found, starting to listen..."* e começa a ouvir potenciais novas conexões.
 - 1.2. Caso haja um servidor S_j aberto para conexão, ele recebe a requisição de S_i e verifica se a quantidade máxima de conexões peer-2-peer foi alcançada.
 - 1.2.1. Em caso positivo, o servidor S_j responde uma mensagem de **ERROR(01)** para S_i
 - 1.2.1.1. Servidor S_i recebe a mensagem **ERROR(01)** e imprime na tela a descrição do código de erro correspondente (vide *Especificação das Mensagens*).
 - 1.2.2. Em caso negativo, servidor S_j define um identificador **Pid S_i** para S_i , registra o identificador em sua base de dados, imprime em tela a mensagem *"Peer Pid S_i connected"* e envia para S_i o identificador **Pid S_i** definido através da mensagem **RES_CONNPEER(Pid S_i)**.
 - 1.2.2.1. S_i recebe o seu identificador através da mensagem **RES_CONNPEER(Pid S_i)**, imprime na tela *"New Peer ID: Pid S_i "* define um identificador **Pid S_j** para o servidor S_j , imprime em tela a mensagem *"Peer Pid S_j connected"* e envia para S_j o identificador através da mensagem **RES_CONNPEER(Pid S_j)**
 - 1.2.2.2. S_j recebe o seu identificador através da mensagem **RES_CONNPEER(Pid S_j)**, imprime na tela *"New Peer ID: Pid S_j "*.

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

1. O servidor S_i recebe comando via teclado *"kill"* e solicita ao servidor S_j o fechamento de comunicação através da mensagem **REQ_DISCPEER(Pid S_i)**.
2. Servidor S_j recebe **REQ_DISCPEER(Pid S_i)** e verifica se **Pid S_i** é o identificador do peer conectado.
 - 2.1. Em caso negativo, o servidor S_j responde mensagem de erro **ERROR(02)** para o servidor S_i .
 - 2.1.1. Servidor S_i recebe mensagem de **ERROR(02)** de S_j e imprime na tela a descrição do código de erro correspondente (vide *Especificação das Mensagens*).
 - 2.2. Em caso positivo, o servidor S_j remove S_i da sua base de dados, responde mensagem **OK(01)** para S_i e imprime na tela a mensagem *Peer Pid S_i disconnected*.
 - 2.2.1. Servidor S_i recebe mensagem **OK(01)** de confirmação, imprime em tela a descrição da mensagem, remove S_j da sua base de dados, imprime na tela a mensagem *Peer Pid S_i disconnected* e encerra a sua execução.
 - 2.2.2. Servidor S_j imprime em tela a mensagem *"No peer found, starting to listen..."* e começa a ouvir potenciais novas conexões.

Abertura de comunicação de Cliente com Servidores

1. Um cliente C_i é iniciado e solicita aos servidores **SS** e **SL** a abertura de comunicação a fim de obter seu identificador na rede.
2. Os servidores **SS** e **SL** recebem requisição de C_i e verifica se quantidade máxima de conexões foi alcançada.
 - 2.1. Em caso positivo, os servidores **SS** e **SL** imprimem a mensagem de erro **ERROR(09)** (vide *Especificação das Mensagens*).
 - 2.1.1. Cliente C_i imprime em tela descrição do código de erro **ERROR(09)**.
 - 2.2. Em caso negativo, cliente C_i envia a mensagem **REQ_CONNSEN(ID)** para os servidores **SS** e **SL**.
 - 2.2.1. Os servidores **SS** e **SL** registram o **Id C_i** de C_i único entre os seus clientes, imprime em tela a mensagem *"Client Id C_i added (Loc LocId)"* para o servidor SL e *Client Id C_i added (StatusSens)* para o servidor SS. Após envia para o cliente C_i a mensagem **RES_CONNSEN(Id C_i)**. O cliente C_i , ao receber as mensagens **RES_CONNSEN(Id C_i)** dos servidores **SS** e **SL**, imprime em tela ambas as mensagens:

SS New ID: Id C_i
SL New ID: Id C_i
Ok(02)

- Tanto a geração de localização quanto de status são geradas aleatoriamente dentro dos respectivos servidores.
- A geração do ID deve ser gerada dentro do código do sensor.

Fechamento de comunicação de Cliente com Servidores

1. Um cliente **C_i** (IC) recebe comando via teclado
kill
e solicita aos servidores **SS** e **SL** o fechamento da comunicação por meio da mensagem **REQ_DISCSEN(IdC_i)**.
2. Os servidores **SS** e **SL** recebem **REQ_DISCSEN(IdC_i)** e verificam se **IdC_i** existe na base de dados.
 - 2.1. Em caso negativo, os servidores **SS** e **SL** respondem mensagem de erro **ERROR(10)** para cliente **C_i**.
 - 2.1.1. Cliente **C_i** recebe mensagem **ERROR(10)** dos servidores **SS** e **SL** e imprime em tela a descrição do código de erro correspondente (vide *Especificação das Mensagens*).
 - 2.2. Em caso positivo, os servidores **SS** e **SL** removem **C_i** da base de dados, respondem mensagem **OK(01)** para **C_i**, desconectam **C_i**, e imprimem em tela a mensagem
Client IdC_i removed (Loc LocId)
 - 2.2.1. Cliente **C_i** recebe mensagem **OK(01)** de confirmação, imprime em tela as mensagens
SS Successful disconnect
SL Successful disconnect
para a mensagens vinda do servidor **SS** e **SL**, respectivamente, fecha as conexões e encerra a sua execução.

Descrição das funcionalidades

Esta seção descreve o fluxo de mensagens transmitidas entre os clientes (**ICs**) e os servidores (**SS** e **SL**) resultante de cada uma das quatro funcionalidades da aplicação a fim de monitorar e controlar o acesso das pessoas aos prédios do campus.

1)Alerta de pane elétrica detectada por sensor (Ci → SS → SL → SS → Ci)

1. Um cliente sensor **C_i** recebe um comando de check de falha:

Check failure

O cliente envia a mensagem **REQ_SENSSTATUS(StatId)** para o servidor de status **SS**, informando o ID do sensor.

※ O servidor **SS** recebe a mensagem e imprime:

REQ_SENSSTATUS *SensId*

2. O servidor *SS* verifica, em sua base de dados, se o sensor com ID *SensID* está com **status 1 (falha)** ou **0 (normal)**.

- a. Se o status for **0**, nenhuma ação adicional é tomada.
Responde mensagem **OK(03)**
- b. Se o status for **1**, o servidor *SS* envia a mensagem
REQ_CHECKALERT(SensId) ao servidor de localização *SL*

※ O servidor *SL* recebe a mensagem e imprime:

REQ_CHECKALERT StatId

3. O servidor *SL* verifica a localização do sensor com ID *SensID* em sua base de dados:

- a. Se **não encontrar** a localização do sensor, responde com a mensagem de erro *ERROR(10)* para o servidor *SS*.
 - i. O servidor *SS* então envia esta mensagem ao cliente *Ci*, que imprime:
1. *Sensor not found*
- b. Se **encontrar** a localização, o servidor *SL* envia a mensagem
RES_CHECKALERT(LocId) para o servidor *SS*.

4. O servidor **SS** recebe a mensagem e envia para o cliente a mensagem:
RES_SENSSTATUS(LocID)

5. O cliente imprime a localização da falha, incluindo a **região correspondente**:

- Se **location = 1,2 e 3** imprime: *Alert received from area: 1 (Norte)*
- Se **location = 4 e 5**, imprime: *Alert received from area: 2 (Sul)*
- Se **location = 6 e 7**, imprime: *Alert received from area: 3 (Leste)*
- Se **location = 8,9 e 10**, imprime: *Alert received from area: 4 (Oeste)*

2) Consultar informações de localização de um sensor (Ci -> SL)

1. Um cliente Ci recebe o comando via teclado:

`locate SensID`

2. O cliente envia a mensagem `REQ_SENSLOC(SensID)` ao servidor SL.

※ O servidor SL recebe e imprime:

`REQ_SENSLOC SensID`

3. O servidor SL verifica se o sensor com `SensID` está registrado.
4. Em caso negativo, responde a mensagem `ERROR(10)` para o cliente Ci.
5. O cliente imprime:

`Sensor not found`

6. Em caso positivo, o servidor SL responde com `RES_SENSLOC(LocId)`.
7. O cliente imprime:

`Current sensor location: LocId`

3) Diagnóstico da condição de sensores em uma localização (Ci → SL)

1. Um cliente Ci recebe o comando via teclado:

`diagnose LocId`

2. O cliente envia a mensagem `REQ_LOCLIST(LocId)` ao servidor SL.

※ O servidor SL imprime:

`REQ_LOCLIST LocId`

3. O servidor SL busca os sensores cadastrados em LocId.
4. Em caso de erro ou localização inválida, responde `ERROR(10)`.
O cliente imprime:

`Location not found`

5. Se for bem-sucedido, responde com `RES_LOCLIST(SenIDs)` contendo os sensores da localização.
O cliente imprime:

`Sensors at location LocId: SenID1, SenID2, SenID3`

Exemplos de saídas:

Consulta de sensores ativos em uma área

1. [CLIENT] check failure

[CLIENT] Sending REQ_SENSSTATUS 0000123456

[SS] REQ_SENSSTATUS 0000123456

[SS] Sensor 0000123456 status = 1 (failure detected)

[SS] Sending REQ_CHECKALERT 0000123456 to SL

[SL] REQ_CHECKALERT 0000123456
[SL] Found location of sensor 0000123456: location 3
[SL] Sending RES_CHECKALERT 3 to SS

[SS] RES_CHECKALERT 3
[SS] Sending RES_SENSSTATUS 3 to CLIENT

[CLIENT] Alert received from area: 1 (Norte)

2. [CLIENT] check failure
[CLIENT] Sending REQ_SENSSTATUS 0000999999

[SS] REQ_SENSSTATUS 0000999999
[SS] Sensor 0000999999 status = 1 (failure detected)
[SS] Sending REQ_CHECKALERT 0000999999 to SL

[SL] REQ_CHECKALERT 0000999999
[SL] ERROR(10) - Sensor not found

[SS] ERROR(10) received from SL
[SS] Sending ERROR(10) to CLIENT

[CLIENT] Sensor not found

Diagnóstico de sensores em uma localização:

1. [CLIENT] diagnose 02
[CLIENT] Sending REQ_LOCLIST 02
[SL] REQ_LOCLIST 02
[SL] Found sensors at location 02
[SL] Sending RES_LOCLIST 0000123456, 0000111122,0000998877
[CLIENT] Sensors at location 02: 0000123456, 0000111122, 0000998877

[CLIENT] diagnose 07
[CLIENT] Sending REQ_LOCLIST 07
[SL] REQ_LOCLIST 07
[SL] Location 07 not found
[SL] Sending ERROR 10
[CLIENT] Location not found

IMPLEMENTAÇÃO

Pequenos 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 TCP. Haverá um socket em cada **equipamento**, 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. A medida que vai se conectando e se desconectando de clientes, outros sockets são adicionados/descartados do seu pool de sockets.

O tipo de endereço utilizado neste trabalho prático deve ser IPv4. Um número máximo de **2 servidores serão executados simultaneamente**. Cada **servidor** deve tratar até **15 equipamentos simultaneamente**. Ademais, o servidor é responsável por definir identificações únicas para cada equipamento na rede. Um **equipamento (cliente)** inicia sem identificação, após a solicitação de entrada na rede ele recebe sua identificação. Também, o equipamento e o servidor devem receber mensagens do teclado.

Comandos

- **Servidor**

O servidor deve receber alguns comandos pela entrada padrão (teclado) a fim de interagir com o sistema. Tais comandos são descritos a seguir:

- **close connection:** Comando requisita o fechamento de comunicação de um peer com o outro caso haja essa conexão. Caso não haja conexão, o programa deverá imprimir *No peer connected to close connection*

- **Sensor**

O equipamento deve receber alguns comandos pela entrada padrão (teclado) a fim de interagir com o sistema. Tais comando são descritos a seguir:

- **close connection :** Comando requisita fechamento de comunicação, o que dispara o fluxo de mensagem de controle "Fechamento de comunicação com Servidor".
- **Check failure:** Lista area com falha

→ **diagnose LocId**: Diagnóstico da condição de sensores em uma localização

→ **locate SensID**: Consultar informações de localização de um sensor

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 (Sugestão: utilize a porta 64000 para a conexão peer-2-peer e as portas 60000 e 61000 para a conectar com clientes para efeitos de padronização do trabalho). Seu equipamento (cliente) deve receber, **estritamente nessa ordem**, o endereço IP e as respectivas portas dos respectivos servidores em que ele deseja se conectar para estabelecimento da comunicação. Para realizar múltiplas conexões de equipamentos com o servidor basta executar múltiplas vezes o código do equipamento.

A seguir, um exemplo de execução de dois equipamentos conectados com um servidor em três terminais distintos:

```
Terminal 1: ./server 127.0.0.1 64000 60000
Terminal 2: ./server 127.0.0.1 64000 61000
Terminal 3: ./sensor 127.0.0.1 60000 61000
Terminal 4: ./sensor 127.0.0.1 61000 60000
```

As portas são definidas da seguinte forma:

```
./server <IP> <porta_peer2peer> <porta_clientes>
./sensor <IP> <porta_servidor1> <porta_servidor2>
```

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 do seu servidor na tela, e.g., para depuração, não impacta os resultados dos testes).

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

- **Cliente-Servidor : 40% dos pontos**
 - Abertura de comunicação:: **10% dos pontos**
 - Fechamento de comunicação: **10% dos pontos**
 - Requisição de informação de um equipamento: **10% dos pontos**
 - Funcionalidades de equipamentos: **20% dos pontos**
- **Servidor-Servidor (Peer-2-Peer): 40% dos pontos**
 - Abertura de comunicação:: **10% dos pontos**
 - Fechamento de comunicação: **10% dos pontos**
 - Requisição de informação de um equipamento: **10% dos pontos**

Total: **80% dos pontos** + 20% dos pontos (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, equipamento, discussão e conclusão. O relatório deve ser entregue em formato PDF. A documentação corresponde a 20% dos pontos do trabalho (**+8 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 “sensor” 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

- sensor.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 [TP-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 identado.
- 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 cliente.

Exemplos de execução

Exemplos de execução serão disponibilizados brevemente.