Arquitetura de Monitoramento de Falhas em Redes Elétricas (v1.0)

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 sensoriamento 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

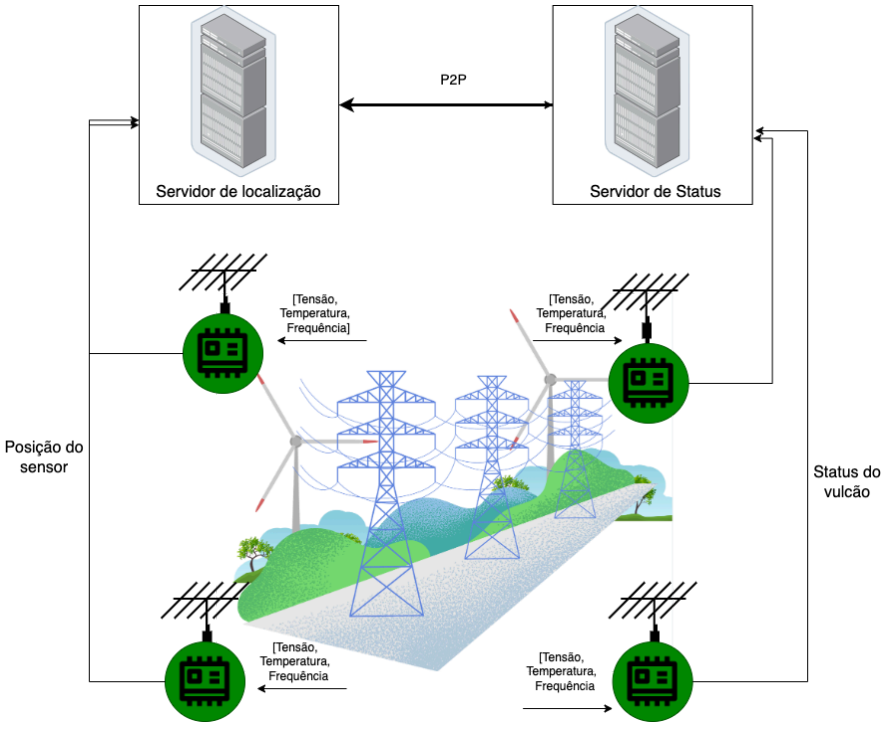

Figura 1 – Exemplo de cenário da arquitetura de monitoramento de falhas em uma rede elétrica, ilustrando a coleta de informações por sensores e servidores que integram a infraestrutura.

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.

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 dividas 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	21	PidS_i	Mensagem de resposta de conexão entre

(S_j -> S_i)			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	LocId	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_STATALERT (SL -> SU)	36	StatId	Mensagem de requisição de alerta de status positivo de pane na rede contendo o ID do sensor
RES_STATALERT	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	UID	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_LOCLIST (C_i -> SL)	40	SenID, LocId	Mensagem de requisição de status de um sensor presentes em uma localização
RES_LOCLIST (SL -> C_i)	41	SenIDs	Mensagem de resposta de status sobre um sensor presente em uma localização
REQ_AREALOG (Si → SM)	42	SensorId, AreaId	Requisição da lista de sensores ativos em uma área

Mensagens de Erro ou Confirmação			
Tipo	**Código**	**Payload**	**Descrição**
ERROR	255	Code	Mensagem de erro transmitida do Servidor para sensor Sen_i. O campo payload informa o código de erro. Abaixo descrição de cada código: 01: "*Peer limit exceeded*" 02: "*Peer not found*" 09: "*Sensor limit exceeded*" 10: *"Sensor not found"*
OK	0	Code	Mensagem de confirmação transmitida do Servidor para sensor Sen_i. O campo payload informa o código de confirmação. Abaixo descrição de cada código: 01: "*Successful disconnect*" 02: "*Successful create*" 03: "*Successful update*"

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 `Si` tenta se conectar ao servidor `Sj` enviando a mensagem `REQ_CONNPEER()`
   - 1.1. Caso não haja um servidor $\mathbf{S_j}$ aberto para conexão, o servidor $\mathbf{S_i}$ imprime em tela a mensagem *"No peer found, starting to listen..."* e começa a ouvir potenciais novas conexões.
   - 1.2. Caso haja um servidor $\mathbf{S_j}$ aberto para conexão, ele recebe a requisição de $\mathbf{S_i}$ e verifica se a quantidade máxima de conexões peer-2-peer foi alcançada.
     - 1.2.1. Em caso positivo, o servidor $\mathbf{S_j}$ responde uma mensagem de **ERROR(01)** para $\mathbf{S_i}$
       - 1.2.1.1. Servidor $\mathbf{S_i}$ 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 $\mathbf{S_j}$ define um identificador $\mathbf{PidS_i}$ para $\mathbf{S_i}$, registra o identificador em sua base de dados, imprime em tela a

mensagem *"Peer $PidS_i$ connected"* e envia para $\mathbf{S_i}$ o identificador $\mathbf{PidS_i}$ definido através da mensagem **RES_CONNPEER($\mathbf{PidS_i}$).**

1.2.2.1. $\mathbf{S_i}$ recebe o seu identificador através da mensagem **RES_CONNPEER($\mathbf{PidS_i}$),** imprime na tela *"New Peer ID: $PidS_i$"* define um identificador $\mathbf{PidS_j}$ para o servidor $\mathbf{S_j}$, imprime em tela a mensagem *"Peer $PidS_j$ connected"* e envia para $\mathbf{S_j}$ o identificador através da mensagem **RES_CONNPEER($\mathbf{PidS_j}$)**

1.2.2.2. $\mathbf{S_j}$ recebe o seu identificador através da mensagem **RES_CONNPEER($\mathbf{PidS_j}$),** imprime na tela *"New Peer ID: $PidS_j$"*.

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

1. O **servidor $\mathbf{S_i}$** recebe comando via teclado "*kill*"
   e solicita ao **servidor $\mathbf{S_j}$** o fechamento de comunicação através da mensagem **REQ_DISCPEER($\mathbf{PidS_i}$)**.
2. **Servidor $\mathbf{S_j}$** recebe **REQ_DISCPEER($\mathbf{PidS_i}$)** e verifica se $\mathbf{PidS_i}$ é o identificador do peer conectado.
   2.1. Em caso negativo, o servidor $\mathbf{S_j}$ responde mensagem de erro **ERROR(02)** para o servidor $\mathbf{S_i}$.
      2.1.1. Servidor $\mathbf{S_i}$ recebe mensagem de **ERROR(02)** de $\mathbf{S_j}$ e imprime na tela a descrição do código de erro correspondente (vide *Especificação das Mensagens)*.
   2.2. Em caso positivo, o servidor $\mathbf{S_j}$ remove $\mathbf{S_i}$ da sua base de dados, responde mensagem **OK(01)** para $\mathbf{S_i}$ e imprime na tela a mensagem *Peer $PidS_i$ disconnected*.
      2.2.1. **Servidor $\mathbf{S_i}$** recebe mensagem **OK(01)** de confirmação, imprime em tela a descrição da mensagem, remove $\mathbf{S_j}$ da sua base de dados, imprime na tela a mensagem *Peer $PidS_j$ disconnected* e encerra a sua execução.
      2.2.2. **Servidor $\mathbf{S_j}$** 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 $\mathbf{C_i}$ é iniciado para uma localização **LocId** e verifica se **LocId** é um valor entre **1 e 10**, inclusive.
   1.1. Caso negativo, $\mathbf{C_i}$ imprime na tela a mensagem *Invalid argument* e encerra a sua execução sem trocar nenhuma mensagem na rede.
   1.2. Caso positivo, $\mathbf{C_i}$ continua a execução de acordo com o passo 2.
2. O cliente $\mathbf{C_i}$ solicita aos servidores **SS** e **SL** a abertura de comunicação a fim de obter seu identificador na rede.
3. Os servidores **SS** e **SL** recebem requisição de $\mathbf{C_i}$ e verifica se quantidade máxima de conexões foi alcançada.

3.1. Em caso positivo, os servidores **SS** e **SL** imprimem a mensagem de erro **ERROR(09)** (vide *Especificação das Mensagens*).

3.1.1. Cliente $\mathbf{C_i}$ imprime em tela descrição do código de erro **ERROR(09)**.

3.2. Em caso negativo, cliente $\mathbf{C_i}$ envia a mensagem **REQ_CONNSEN(LocId)** para os servidores **SS** e **SL.**

3.2.1. Os servidores **SS** e **SL** definem um identificador $\mathbf{IdC_i}$ para $\mathbf{C_i}$ único entre os seus clientes, registra o identificador em sua base de dados, imprime em tela a mensagem "*Client* IdC_i *added (Loc* ***LocId****)*" e envia para o cliente $\mathbf{C_i}$ a mensagem **RES_CONNSEN(**$\mathbf{IdC_i}$**).** O cliente $\mathbf{C_i}$, ao receber as mensagens **RES_CONNSEN(**$\mathbf{IdC_i}$**)** dos servidores **SS** e **SL**, registra sua nova identificação e imprime em tela ambas as mensagens

SS New ID: $\mathbf{IdC_i}$
SL New ID: $\mathbf{IdC_i}$

Fechamento de comunicação de Cliente com Servidores

1. Um cliente $\mathbf{C_i}$ (IC) recebe comando via teclado

   kill

   e solicita aos servidores **SS** e **SL** o fechamento da comunicação por meio da mensagem **REQ_DISCSEN(**$\mathbf{IdC_i}$**)**.
2. Os servidores **SS** e **SL** recebem **REQ_DISCSEN(**$\mathbf{IdC_i}$**)** e verificam se $\mathbf{IdC_i}$ existe na base de dados.

   2.1. Em caso negativo, os servidores **SS** e **SL** respondem mensagem de erro **ERROR(10)** para cliente $\mathbf{C_i}$.

   2.1.1. Cliente $\mathbf{C_i}$ 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 $\mathbf{C_i}$ da base de dados, respondem mensagem **OK(01)** para $\mathbf{C_i}$, desconectam $\mathbf{C_i}$, e imprimem em tela a mensagem

   Client $\mathbf{IdC_i}$ *removed (Loc* ***LocId****)*

   2.2.1. Cliente $\mathbf{C_i}$ 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 → SL → SU)

1. Um cliente sensor Ci detecta uma falha e inicia um alerta

   O cliente envia a mensagem `REQ_STATALERT(StatId)` para o servidor de localização SL, informando o ID do sensor que detectou a falha.

   ※ O servidor SL recebe a mensagem e imprime:

   `REQ_STATALERT StatId`
2. O servidor SL verifica a localização do sensor com ID StatId em sua base de dados.

   Se não encontrar a localização do sensor, responde a mensagem de erro `ERROR(10)` para o sensor Ci.

   O cliente Ci imprime:

   `Sensor not found`
3. Se encontrar a localização, o servidor SL envia a mensagem `RES_STATALERT(LocId)` para o servidor SU.
4. O servidor **SS** recebe a mensagem e imprime a localização da falha, incluindo a **região correspondente**:
   - Se **LocId = 1**, imprime: `Alert received from location: 1 (Norte)`
   - Se **LocId = 2**, imprime: `Alert received from location: 2 (Sul)`
   - Se **LocId = 3**, imprime: `Alert received from location: 3 (Leste)`
   - Se **LocId = 4**, imprime: `Alert received from location: 4 (Oeste)`

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

1. Um cliente Ci recebe o comando via teclado:

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

   ※ O servidor SL recebe e imprime:

   `REQ_SENSLOC UID`
3. O servidor SL verifica se o sensor com UID 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) Consulta de sensores ativos em uma área (Ci → SL)

1. Um cliente Ci recebe o comando via teclado:
   `list AreaId`
2. O cliente envia a mensagem `REQ_AREALOG(SensorId, AreaId)` para o servidor SL.
   ※ O servidor SL imprime:
   `REQ_AREALOG SensorId AreaId`
3. O servidor SL busca em sua base os sensores ativos da área AreaId.
4. O servidor responde ao cliente Ci com a lista formatada em `RES_LOCLIST(SenIDs)`.
   4.1. O cliente imprime:
   `Active sensors in Area AreaId: SenID1, SenID2, SenID3`
5. (Separados por vírgula e espaço; lista pode estar vazia.)

4) 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(SenID, LocId)` ao servidor SL.
   ※ O servidor SL imprime:
   `REQ_LOCLIST SenID 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] Sending REQ_STATALERT 0000123456
   [SL] REQ_STATALERT 0000123456
   [SS] Alert received from location: 03
2. [CLIENT] Sending REQ_SENSLOC 0000123456
   [SL] REQ_SENSLOC 0000123456
   [CLIENT] Current sensor location: 05

Diagnóstico de sensores em uma localização:

1. [CLIENT] Sending REQ_AREALOG 0000001111 07
   [SL] REQ_AREALOG 0000001111 07
   [CLIENT] Active sensors in Area 07: 0000123456, 0000111122, 0000998877
2. [CLIENT] Sending REQ_LOCLIST 0000001111 02
   [SL] REQ_LOCLIST 0000001111 02
   [CLIENT] Sensors at location 02: 0000123456, 0000654321

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*
- ➔ **list equipment** : Comando lista os equipamentos na base da dados do equipamento por id, separando equipamentos por cluster.

- **Equipamento**

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”.
- ➔ **list equipment** : Comando lista os equipamentos na base da dados do equipamento por id.
- ➔ **request information from IdEq$_j$:** Comando requisita informação de equipamento **IdEq$_j$** o que dispara o fluxo de mensagem de dados “Equipamento Eq$_i$ solicita informação para equipamento Eq$_j$ na rede”

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 a segunda porta do fornecida ao servidor 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 2: ./equipment 127.0.0.1 60000
Terminal 3: ./equipment 127.0.0.1 61000
```

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**
  - Listagem de equipamentos: **10% 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**
  - Listagem de equipamentos: **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 "equipment" 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
    - equipment.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.