NetLab — Telemetria Confiável em Redes

Projeto avaliativo da disciplina Redes de Computadores — Engenharia de Software

1. Apresentação

Este documento especifica o projeto avaliativo da disciplina Redes de Computadores. O projeto acompanha o semestre e transforma os conceitos de transporte, protocolos de aplicação, resolução de nomes, observabilidade, testes e diagnóstico em um pequeno serviço de telemetria escrito em C++23.

O projeto recebe o nome NetLab — Telemetria Confiável em Redes. “NetLab” indica que a aplicação funciona como laboratório controlado de comunicação; o complemento explicita a competência central: construir, observar, corrigir e explicar um serviço que opera mesmo quando mensagens atrasam, se perdem ou são duplicadas.

A atividade é dividida em duas fases. A primeira exige um serviço UDP funcional e observável. A segunda reutiliza e corrige a primeira versão, acrescenta confiabilidade na aplicação, implementa uma versão equivalente sobre TCP e exige diagnóstico orientado por evidência. Essa continuidade cria uma oportunidade real de recuperação durante a disciplina: corrigir defeitos demonstrados melhora a parcela recuperável da primeira nota.

2. Identificação do projeto

Campo	Definição
Nome	NetLab — Telemetria Confiável em Redes
Linguagem	C++23
Modalidade	Projeto em equipe, com avaliação individual de autoria
Fase 1	Construir e observar um serviço UDP
Fase 2	Corrigir, tornar confiável, comparar com TCP e diagnosticar
Ambiente	Windows ou Linux; execução local e, quando possível, entre duas máquinas
Interfaces	Linha de comando, arquivos de configuração simples, logs e captura de pacotes

3. Motivação e problema real

Sistemas de software recebem continuamente medições de sensores, agentes, dispositivos e serviços. Temperatura, uso de CPU, latência, contagem de eventos e estado operacional são exemplos de telemetria. Em redes reais, essas informações atravessam enlaces sujeitos a atraso, perda, duplicação, filtragem e configuração incorreta.

Um programa que envia uma mensagem e imprime uma resposta no caso feliz pode ser produzido rapidamente. O problema profissional começa quando não há resposta, quando a mesma medição chega duas vezes, quando o valor não pode ser convertido ou quando o cliente e o servidor discordam sobre o significado de um campo. A equipe precisa reconhecer o que foi observado, o que pode ser concluído e qual teste separa as hipóteses restantes.

O NetLab simula esse contexto em escala compatível com alunos do segundo período. Não há interface gráfica, banco de dados, criptografia, servidor multithread ou implantação obrigatória em nuvem. A complexidade está no contrato da mensagem, na validação, nos testes, nas evidências e no raciocínio sobre falhas.

4. Quem usaria o sistema

- Uma equipe de desenvolvimento que precisa receber telemetria de processos distribuídos e verificar se cada medição foi aceita;
- Uma equipe de operações que investiga por que uma medição não apareceu no servidor;
- Uma equipe de qualidade que testa o comportamento do protocolo diante de entradas inválidas, perda, atraso e duplicação;
- Estudantes de Engenharia de Software que precisam relacionar código de aplicação, transporte, logs e pacotes observados na rede.

5. Casos de uso concretos

- Enviar ao servidor uma medição de temperatura identificada por sensor e número de sequência;
- Rejeitar mensagem truncada, operação desconhecida, identificador inválido, valor não numérico ou sequência fora do intervalo;
- Consultar estatísticas de um sensor: quantidade, mínimo, máximo e média das medições aceitas;
- Registrar em log os eventos necessários para reconstruir uma execução;
- Reenviar uma medição quando a confirmação não chega dentro do prazo, sem registrá-la duas vezes;
- Executar a mesma operação em UDP e TCP e comparar bytes transmitidos, latência e comportamento sob falha;
- Diagnosticar uma perturbação sorteada, distinguindo observação, conclusão, hipóteses e próximo teste.

6. Objetivos de aprendizagem

Ao concluir o projeto, cada estudante deverá ser capaz de:

- Explicar a diferença entre as garantias oferecidas por UDP, TCP e pelo protocolo da aplicação;
- Projetar e documentar um protocolo textual pequeno, com gramática, limites e respostas de erro;
- Tratar dados recebidos pela rede como entrada não confiável;
- Separar transporte, análise da mensagem, armazenamento das medições e diagnóstico;
- Escrever testes automatizados para casos válidos, inválidos e regressões;
- Usar logs e captura de pacotes como evidência, evitando conclusões maiores que a observação;
- Medir e comparar implementações equivalentes sem confundir uma execução isolada com evidência geral;
- Avaliar sugestões de inteligência artificial por sintaxe, semântica e raio de exposição.

7. Regras gerais

7.1 Equipes e autoria

O projeto pode ser realizado em equipes de até quatro estudantes. O código é coletivo; a compreensão é individual. A composição da equipe não muda durante uma fase. Cada integrante deve contribuir de forma identificável no repositório e dominar o sistema entregue.

A autoria será verificada por marcos semanais, histórico de versões, testes ocultos, explicação de trecho sorteado, diagnóstico de uma falha reproduzível e pequena alteração solicitada durante a demonstração. Contribuição não registrada não será considerada na avaliação do processo.

7.2 Repositório e processo

A equipe manterá um repositório Git. O ramo principal conterá somente versões integradas. Mudanças relevantes devem apresentar mensagem descritiva e, quando a plataforma permitir, revisão por outro integrante.

O repositório deverá conter:

- README.md com visão geral, compilação, execução, exemplos e limitações;
- LICENSE com licença escolhida e justificativa;
- .gitignore adequado à cadeia de compilação;
- documentação do protocolo;
- código-fonte, testes e arquivos de configuração de exemplo;
- registro das decisões técnicas e das sugestões de IA avaliadas;
- ausência total de executáveis, credenciais, tokens ou dados pessoais desnecessários.

7.3 Portabilidade

O professor fornecerá uma camada mínima para inicialização e encerramento de sockets em Windows e Linux. A avaliação recai sobre protocolo, validação, observação e diagnóstico. A equipe deve usar CMake ou processo equivalente documentado e manter a implementação compilável em pelo menos um dos ambientes indicados pelo professor.

8. Arquitetura obrigatória

A solução será dividida em componentes com responsabilidades explícitas. Dependências circulares e mistura entre análise de protocolo e chamadas de socket prejudicam a testabilidade e serão tratadas como defeito arquitetural.

- netlab-cliente: recebe parâmetros pela linha de comando, constrói mensagens, envia medições, aguarda respostas e produz métricas;
- netlab-servidor: recebe bytes, delega a análise, aplica regras, armazena estatísticas em memória e responde;
- biblioteca de protocolo: converte texto em estruturas tipadas e estruturas em texto, sem abrir sockets;
- camada de transporte: encapsula operações UDP e TCP necessárias ao projeto;
- armazenamento em memória: mantém medições e controle de duplicatas durante a execução;
- diagnóstico: organiza evidências e gera relatório estruturado;
- netlab-testes: executa testes unitários e de integração sem depender de interação manual.

A equipe pode organizar arquivos e namespaces de outra maneira, desde que preserve essas separações e seja capaz de demonstrá-las.

9. Protocolo da aplicação

9.1 Mensagem de medição

O formato textual obrigatório usa uma linha UTF-8 terminada por quebra de linha. A forma canônica é:
MEDIR|<sensor>|<tipo>|<valor>|<sequencia>
Exemplo: MEDIR|estufa-07|temperatura|23.50|1042

9.2 Respostas

- ACK|<sequencia> — medição válida aceita;
- DUP|<sequencia> — sequência já processada para o mesmo sensor;
- ERR|SINTAXE|<detalhe> — mensagem cuja forma não pode ser analisada;
- ERR|SEMANTICA|<detalhe> — forma válida, mas operação, tipo, valor ou limite inválido;
- ESTAT|<sensor>|<quantidade>|<minimo>|<maximo>|<media> — resposta à consulta de estatísticas.

9.3 Regras mínimas

- A mensagem completa, incluindo a quebra de linha, terá no máximo 256 bytes;
- sensor conterá de 1 a 32 caracteres alfanuméricos, hífen ou sublinhado;
- tipo será temperatura, umidade, latencia ou cpu, salvo parâmetro específico fornecido à equipe;
- valor será decimal finito e deverá respeitar a faixa definida para o tipo; NaN e infinito são inválidos;
- sequencia será inteiro sem sinal de 32 bits representado em decimal;
- campos extras, ausentes ou vazios serão rejeitados;
- o analisador nunca deve acessar memória fora dos limites nem encerrar o processo por entrada inválida.

9.4 Parâmetros por equipe

O professor poderá atribuir a cada equipe porta padrão, prefixo de identificador, tipos adicionais, faixas de valor, tempo limite e quantidade máxima de tentativas. Esses parâmetros tornam as entregas distinguíveis e serão usados nos testes ocultos.

10. Fase 1 — Construir e observar

A primeira fase começa após o conteúdo de telemetria e diagnóstico. O objetivo é produzir um serviço UDP pequeno, correto no contrato declarado e suficientemente observável para explicar uma falha.

10.1 Requisitos funcionais

- Resolver o nome ou endereço do servidor informado pela linha de comando;
- Enviar medições por UDP e apresentar a resposta recebida;
- Receber datagramas sem supor que o conteúdo é válido ou terminado corretamente;

- Analisar e validar todos os campos antes de alterar o estado do servidor;
- Registrar somente medições válidas;
- Calcular quantidade, mínimo, máximo e média por sensor e tipo;
- Responder a consultas de estatística previstas na especificação da equipe;
- Produzir log estruturado com instante, direção, origem/destino, operação, sequência e resultado;
- Executar testes automatizados do analisador e das regras de validação.

10.2 Evidência de rede

A equipe deverá produzir uma captura curta no Wireshark contendo ao menos uma medição aceita, uma mensagem rejeitada e uma consulta. A captura será acompanhada de comentários que identifiquem endereços, portas, tamanho da mensagem, conteúdo no nível da aplicação e correspondência com o log.

10.3 Entregáveis

- Código-fonte compilável e instruções de execução;
- Especificação do protocolo com no máximo três páginas;
- Testes automatizados do analisador;
- Captura curta comentada no Wireshark;
- Log da execução demonstrada;
- Demonstração de cinco minutos;
- Relato individual de uma página sobre uma decisão de implementação.

10.4 Rubrica da Fase 1

Critério	Peso	Evidência principal
Funcionamento básico	30%	Cliente, servidor e consulta executam conforme o protocolo.
Análise e validação	20%	Entradas inválidas são rejeitadas sem alterar o estado.
Tratamento de falhas	15%	Erros de rede e de entrada são comunicados de forma útil.
Testes	15%	Casos válidos, inválidos e limites são automatizados.
Captura e interpretação	10%	Pacotes, logs e explicação são coerentes.
Defesa individual	10%	Integrante explica decisão e executa o sistema.

11. Devolutiva e recuperação

Depois da primeira entrega, o professor devolverá um relatório por critérios, associado a casos de teste reproduzíveis. Cada defeito corrigível deverá ser transformado pela equipe em teste de regressão antes da correção. A versão originalmente entregue permanecerá identificável no histórico.

A recuperação não é uma repetição da Fase 1. Na Fase 2, os critérios corrigíveis serão reavaliados dentro do projeto ampliado. Corrigir a causa, preservar o teste e demonstrar que não houve regressão caracteriza aprendizagem observável. Esconder o sintoma, alterar o teste para fazê-lo passar ou apagar a primeira versão não caracteriza recuperação.

12. Fase 2 — Tornar confiável e comparar

A segunda fase parte obrigatoriamente da primeira. A equipe deverá corrigir os defeitos apontados, acrescentar confiabilidade limitada sobre UDP, implementar uma operação equivalente sobre TCP, medir as duas alternativas e diagnosticar uma falha sorteada.

12.1 Confirmação e retransmissão

- O cliente associa cada medição a uma sequência e aguarda ACK correspondente;
- A espera usa std::chrono::steady_clock;
- O tempo limite e o número máximo de tentativas são configuráveis dentro dos limites definidos;
- Uma resposta com sequência diferente não confirma a medição esperada;
- Ao esgotar as tentativas, o cliente encerra a operação com erro explícito e métrica correspondente.

12.2 Descarte de duplicatas

- O servidor identifica duplicata pelo par sensor e sequência;

- Uma duplicata recebe resposta DUP ou ACK conforme decisão documentada, mas não altera as estatísticas;
- A estrutura de deduplicação terá limite de memória documentado;
- A equipe explicará o que ocorre quando o número de sequência retorna a zero.

12.3 Versão TCP equivalente

- Cliente e servidor TCP executarão a mesma operação MEDIR e usarão o mesmo conteúdo de aplicação;
- A implementação tratará leitura e escrita parcial;
- A delimitação da mensagem não dependerá de uma chamada recv corresponder a uma mensagem;
- A comparação não atribuirá ao TCP propriedades que pertencem à aplicação, como validação semântica ou idempotência.

12.4 Métricas e comparação

- Número de medições solicitadas, aceitas, duplicadas e não confirmadas;
- Tentativas e confirmações transmitidas;
- Bytes da aplicação e estimativa dos bytes no fio, com premissas declaradas;
- Latência de conclusão por operação e resumo com média e percentis definidos;
- Comparação entre UDP simples, UDP confiável do projeto e TCP sob o mesmo cenário.

12.5 Diagnóstico orientado por evidência

O comando ou relatório diagnosticar deverá separar quatro campos: observação; conclusão sustentada; hipóteses ainda compatíveis; próximo teste. O professor sorteará uma perturbação reproduzível em nome, endereço, porta, formato, atraso, perda, confirmação ou duplicação. A equipe alterará uma variável por teste e explicará como cada evidência reduz o conjunto de hipóteses.

12.6 Entregáveis

- Código corrigido e ampliado, com identificação da versão da Fase 1;
- Testes de regressão correspondentes à devolutiva;
- Testes de confirmação, timeout, retransmissão e duplicatas;
- Implementação TCP equivalente;
- Relatório de comparação com método, cenário, resultados e limites;
- Relatório de diagnóstico da falha sorteada;
- Demonstração final e alteração individual ao vivo.

12.7 Rubrica técnica da Fase 2

Critério	Peso	Evidência principal
Correções da Fase 1	20%	Defeitos viraram regressões e foram corrigidos.
Confiabilidade	25%	ACK, timeout, tentativas e deduplicação preservam o contrato.
Medições e comparação	15%	Cenário equivalente, números reproduzíveis e premissas explícitas.
Diagnóstico	20%	Conclusões proporcionais às evidências e próximo teste adequado.
Testes e regressões	20%	Cobertura dos modos de falha e ausência de regressões.

13. Requisitos não funcionais

- Correção: nenhuma entrada inválida altera estatísticas ou causa comportamento indefinido;
- Portabilidade: compilação documentada em ambiente indicado pelo professor;
- Reprodutibilidade: testes registram parâmetros e, quando houver aleatoriedade, a semente usada;
- Observabilidade: mensagens de erro e logs permitem relacionar evento de aplicação, transporte e teste;
- Desempenho: o servidor processa a carga de teste definida sem crescimento de memória não explicado;
- Usabilidade da linha de comando: --help, erros de argumento claros e códigos de saída coerentes;
- Qualidade: advertências altas do compilador, ausência de vazamentos detectáveis e tratamento explícito de falhas;

- Documentação: comandos apresentados devem ser copiáveis e produzir o resultado descrito.

14. Cronograma e marcos

Momento	Marco	Evidência
Após telemetria	Lançamento	Equipe, repositório, parâmetros e desenho do protocolo.
Semana seguinte	Marco 1	Analisador com testes de mensagens válidas e inválidas.
Antes da Fase 1	Marco 2	Cliente e servidor UDP trocam mensagem e geram log.
Metade do semestre	Entrega Fase 1	Artefatos completos e demonstração curta.
Após devolutiva	Marco 3	Regressões reproduzem os defeitos apontados.
Durante a Fase 2	Marco 4	ACK, timeout, deduplicação e TCP em desenvolvimento.
Final do semestre	Entrega Fase 2	Comparação, diagnóstico e demonstração individual.

15. Nota, devolutiva e recuperação

A nota final do projeto é calculada por $N = 0{,}30 \times F1^* + 0{,}50 \times F2 + 0{,}20 \times D$, em que F1 é a nota da primeira entrega, F2 é a nota técnica da segunda fase e D é a nota individual da demonstração final.

A parcela recuperada da primeira fase é $F1^* = \text{máximo}(F1; 0{,}30 \times F1 + 0{,}70 \times C)$. O valor C reavalia, na Fase 2, apenas os critérios corrigíveis da primeira entrega. A função máximo garante que a recuperação não reduza a nota já obtida.

Exemplo: para F1 = 5,0, C = 8,0, F2 = 7,5 e D = 6,0, a parcela corrigida é $0{,}30 \times 5{,}0 + 0{,}70 \times 8{,}0 = 7{,}10$. Logo, $F1^* = 7{,}10$ e $N = 0{,}30 \times 7{,}10 + 0{,}50 \times 7{,}50 + 0{,}20 \times 6{,}00 = 7{,}08$.

A recuperação exige entrega identificável da Fase 1. Se ela não for entregue, F1 = C = 0. A ausência de uma fase, de artefatos essenciais ou da defesa será tratada conforme as regras institucionais da disciplina.

16. Papel da inteligência artificial

Ferramentas de inteligência artificial são permitidas como apoio. Toda sugestão incorporada que afete o comportamento do sistema deve registrar o teste ou a evidência que a confirmou. O registro mínimo contém tarefa delegada, sugestão recebida, risco percebido, teste executado e decisão de aceitar, adaptar ou rejeitar.

16.1 Permitido

- Explicar mensagens de compilador e documentação de APIs;
- Sugerir organização inicial do CMake, comandos de compilação e texto de documentação;
- Propor casos de teste que a equipe revise e implemente;
- Revisar clareza de logs, mensagens de erro e relatório;
- Gerar uma hipótese de diagnóstico, desde que confrontada com evidência.

16.2 Desaconselhável

- Gerar integralmente o analisador, a máquina de retransmissão ou a deduplicação;
- Aceitar código de sockets sem compreender limites de buffer, retorno parcial e tempo de vida dos dados;
- Usar uma explicação de captura sem relacioná-la aos pacotes e logs efetivamente produzidos.

16.3 Proibido

- Entregar código que nenhum integrante consegue explicar ou alterar;
- Usar biblioteca que implemente o protocolo, a confiabilidade ou o diagnóstico no lugar da equipe;
- Fabricar resultados, capturas, logs, testes ou referências;
- Consultar IA durante a prova de autoria ou a alteração individual ao vivo.

16.4 Verificador em três camadas

- Sintaxe: compila, executa e produz mensagens na forma declarada?
- Semântica: preserva o contrato sob perda, erro, atraso e duplicação?
- Raio de exposição: se a sugestão estiver errada, que estado, medição, entrega ou diagnóstico poderá ser corrompido?

17. Testes públicos e ocultos

Os testes públicos orientarão a implementação, mas não esgotarão a especificação. Os testes ocultos usarão os mesmos contratos e variarão valores, limites, ordem, perda, atraso, duplicação e divisão de leituras TCP. Memorizar exemplos não será suficiente.

- Mensagens vazias, truncadas, acima do limite e com campos extras;
- Valores nos limites e imediatamente fora deles;
- Sequências repetidas, fora de ordem e próximas do retorno a zero;
- Perda da medição, perda da confirmação e confirmação atrasada;
- Leituras TCP fragmentadas e múltiplas mensagens na mesma leitura;
- Nome válido com porta errada, serviço parado e filtro simulado;
- Parâmetros específicos de cada equipe.

18. Prova de autoria

A prova ocorre depois da avaliação técnica. Cada integrante deverá estar preparado para executar, explicar e alterar o projeto. O professor poderá selecionar um trecho, um caso de teste ou uma falha sem aviso prévio.

- Explique por que enviar um datagrama UDP com sucesso não prova que o servidor recebeu a mensagem;
- Mostre onde a entrada deixa de ser bytes e passa a ser uma estrutura validada;
- Demonstre que a perda de um ACK não registra a medição duas vezes;
- Explique por que uma chamada recv em TCP não corresponde necessariamente a uma mensagem;
- Mostre um teste que falhava após a Fase 1 e passou depois da correção;
- Diante de um silêncio, separe observação, hipóteses e próximo teste;
- Implemente uma pequena mudança, como novo tipo de medição ou novo limite, preservando os testes.

A nota D é individual. A incapacidade de explicar ou alterar o sistema reduz somente a parcela individual, salvo quando revelar que a entrega coletiva não satisfaz o contrato técnico.

19. Riscos e considerações éticas

- Telemetria pode expor hábitos, infraestrutura e vulnerabilidades. O projeto usará identificadores fictícios e não coletará dados pessoais reais;
- Logs excessivos podem registrar conteúdo sensível. A equipe deve justificar o que registra e por quanto tempo mantém;
- Uma medição ausente não prova que o sensor falhou. Decisões automáticas devem distinguir ausência de dado de valor normal;
- Retransmissão sem limite pode amplificar congestionamento e indisponibilidade. O projeto exige tentativas limitadas;
- Comparações de desempenho devem declarar ambiente e variabilidade. Resultados isolados não serão apresentados como universais;
- Ferramentas de IA podem produzir código plausível que corrompe estado silenciosamente. A responsabilidade pela verificação permanece com a equipe.

20. Limites deliberados

O NetLab não implementará controle de congestionamento, controle geral de fluxo, ordenação completa de mensagens, criptografia, autenticação forte, persistência em banco, alta disponibilidade, concorrência avançada, interface gráfica, aplicação móvel, Kubernetes ou implantação real em nuvem. Essas exclusões mantêm o projeto compatível com o segundo período e tornam explícito que a confiabilidade construída sobre UDP não o transforma em TCP.

21. Extensões sugeridas

Extensões são opcionais e só serão consideradas depois do cumprimento correto do núcleo. A extensão deve vir acompanhada de teste e justificativa.

- Formato binário adicional com comparação de custo e legibilidade;
- Janela de várias mensagens pendentes, com limite de memória;
- Persistência local segura e recuperação após reinício;
- Exportação de métricas em formato compatível com ferramenta de observabilidade;
- Suporte a IPv6 com os mesmos casos de teste;
- Autenticação simples de mensagens, claramente separada de criptografia completa;
- Visualizador local de linha do tempo construído a partir dos logs;
- Teste de mutação do analisador e relatório dos mutantes sobreviventes.

22. Checklist de entrega

- A versão a avaliar está identificada no ramo principal e por etiqueta;
- O projeto compila a partir de ambiente limpo seguindo o README;
- Todos os testes públicos passam e os resultados estão registrados;
- Não há binários, segredos, dados pessoais ou arquivos temporários versionados;
- A especificação do protocolo corresponde ao código;
- Logs e captura correspondem à mesma execução;
- Métricas informam cenário, unidade e premissas;
- Cada sugestão relevante de IA tem verificação registrada;
- Cada integrante consegue explicar, executar e alterar o sistema;
- Limitações conhecidas estão declaradas com honestidade.

23. Referências e endereços úteis

- IETF. RFC 768: User Datagram Protocol. https://www.rfc-editor.org/rfc/rfc768
- IETF. RFC 9293: Transmission Control Protocol. https://www.rfc-editor.org/rfc/rfc9293
- Microsoft. Winsock documentation. https://learn.microsoft.com/windows/win32/winsock/
- The Open Group. POSIX Sockets. https://pubs.opengroup.org/onlinepubs/9699919799/
- Wireshark Foundation. Wireshark User’s Guide. https://www.wireshark.org/docs/wsug_html_chunked/
- CMake. Documentation. https://cmake.org/documentation/
- Catch2. Documentation. https://github.com/catchorg/Catch2
- GoogleTest. Documentation. https://google.github.io/googletest/