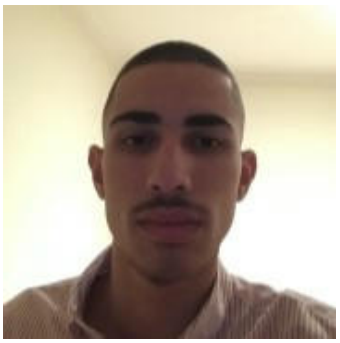
**Universidade do Minho**
Escola de Engenharia
Licenciatura em Engenharia Informática

# Unidade Curricular de Comunicações por Computador

Ano Letivo de 2024/2025

# Relatório Técnico



**Samuel Ferreira**
A100654

**Bruno Campos**
A98639

**Diogo Fernandes**
A100746

December 08, 2024

CC

# Índice

# 1. Introdução

As redes de computadores desempenham um papel essencial em ambientes modernos, permitindo a comunicação contínua e eficiente entre sistemas. No entanto, falhas e degradações de desempenho podem ocorrer devido a problemas de rede, afetando a continuidade dos serviços e a experiência dos utilizadores. Para reduzir esses riscos, a monitorização de redes torna-se fundamental, ao que nos fornece dados em tempo real que tornam possível identificar e solucionar problemas antes que causem impactos significativos.

Este trabalho prático visa desenvolver um sistema de monitorização de redes, denominado Network Monitoring System (NMS), que permite a coleta e análise de métricas essenciais dos dispositivos de rede. A arquitetura deste sistema baseia-se em um modelo distribuído cliente-servidor, composto por dois componentes principais: NMS_Server e NMS_Agent. O NMS_Agent é responsável pela coleta de métricas de rede, tal como o uso de CPU e RAM, jitter e perda de pacotes, bem como o envio dessas informações ao NMS_Server, que centraliza e apresenta esses dados.

Foram, portanto, projetados dois protocolos para a comunicação entre os componentes:

- **NetTask**: Protocolo baseado em UDP para comunicação regular de tarefas e coleta contínua de métricas.
- **AlertFlow**: Protocolo baseado em TCP para notificação de alterações críticas no estado dos dispositivos de rede.

## 2. Arquitetura da solução

Tal como referido anteriormente, o NMS segue uma arquitetura cliente-servidor composta por dois elementos principais: o NMS_Server e o NMS_Agent.

**NMS_Server**: Sendo este o ponto central do sistema, o servidor coordena as operações dos agentes, processando e enviando tarefas descritas em arquivos JSON. Essas tarefas definem as métricas que cada agente deve monitorizar e a frequência de coleta dos dados. O servidor monitoriza quais agentes estão ligados a cada dado momento, e também armazena e apresenta as métricas recebidas e recebe notificações críticas via o protocolo AlertFlow.

**NMS_Agent**: Os agentes são executados em dispositivos de rede (simulados no emulador CORE como hosts) e realizam a coleta de métricas conforme instruído pelo servidor. Utilizando o protocolo NetTask, os agentes enviam periodicamente essas métricas ao servidor, enquanto que o AlertFlow é usado para alertas de eventos críticos.

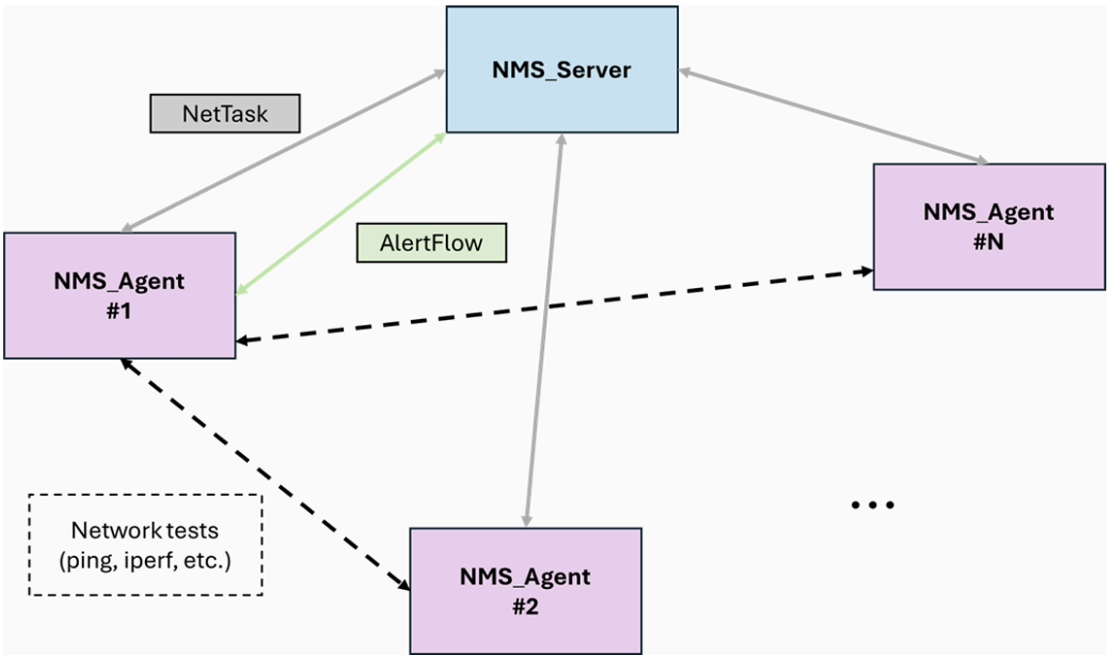



Figura 1: Visão Geral da Arquitetura do NMS

Esta arquitetura permite que múltiplos agentes reportem simultaneamente ao mesmo servidor, tendo como finalidade uma monitorização distribuída e contínua, com baixa sobrecarga de comunicação.

A comunicação é feita com base em ACKs, envio de tarefas, e emissão de métricas e alertas. Esta arquitetura tem suporte para vários agentes em paralelo, usa UDP como protocolo de comunicação e mecanismos simples de confirmação, levando a ser uma arquitetura eficiente, e também é fácil adicionar novas métricas conforme o que seja necessário.

# 3. Especificação dos protocolos usados

## 3.1. Formato das mensagens protocolares

O protocolo é organizado no formato PDU (Unidade de Dados de Protocolo), com campos específicos para garantir que a comunicação entre o servidor e os agentes seja clara e estruturada. Cada campo na PDU tem um propósito bem definido e tamanho fixo ou variável, conforme necessário para melhor eficiência.

**Campos do PDU com respetivos tamanhos e descrições:**

| Tipo | Tamanho | Descrição |
|---|---|---|
| Message_Type | 1 Byte | Tipo da mensagem: Registo, Task, Métrica, Alerta, etc. (ver tabela de tipos abaixo). |
| Seq_Num | 2 Byte | Número de sequência para controlo de retransmissão e ordem |
| Agent_ID | 3 Bytes | Identificador único do agente que enviou a mensagem |
| Task_Type | 1 Byte | Identificador do tipo da tarefa |
| Task_ID | 1 Byte | Identificador universal da tarefa |
| Frequência | 1 Byte | Identifica a que frequência as métricas serão enviadas |
| Duração | 1 Byte | Indica a duração total da tarefa |

**Tipos de Mensagens:**

| Tipo | Tamanho | Descrição |
|---|---|---|
| Registo | 6 bytes | Mensagem de registo do agente no servidor |
| Tarefa | >= 6 bytes | Mensagem com detalhes da tarefa enviada |
| ACK | 3 bytes | Confirmação de recebimento de uma mensagem |
| Fim de Conexão | 2 bytes | Mensagem de encerramento da conexão |

**Mensagem de Registo:**

- Message_Type: 1, 1 byte
- Sequence_Number: 1, 2 bytes
- Agent_ID: “PC1”, 3 bytes

| Message_Type | Agent_ID | Sequence_Number |
|---|---|---|

Tabela 1: PDU referente à mensagem de Registo

**Mensagem de Tarefa:**

- Message_Type: 3 (task), 1 byte
- Agent_ID: “PC1”, 3 bytes
- Sequence_Number: 1, 2 bytes
- Task_ID: 1

- Task_Type: 1
- Informação: “Medir utilização do CPU”

| Msg_Type | Agent_ID | Seq_Number | Task_ID | Task_Type | Data | Freq | Duração |
|---|---|---|---|---|---|---|---|

Tabela 2: PDU referente à mensagem de tarefa

**Mensagem de Acknowledgement:**

- Message_Type: 2
- Agent_ID: “PC1”
- Sequence_Number: 1

| Msg_Type | Agent_ID | Seq_Number |
|---|---|---|

Tabela 3: PDU referente à mensagem de Acknowledgment

**Mensagem de Saída de Agente:**

- Message_Type: 4
-

| Msg_Type | |
|---|---|

Tabela 4: PDU referente à mensagem de envio de métricas e envio de alertas

## 3.2. Diagrama de sequência data troca de mensagens

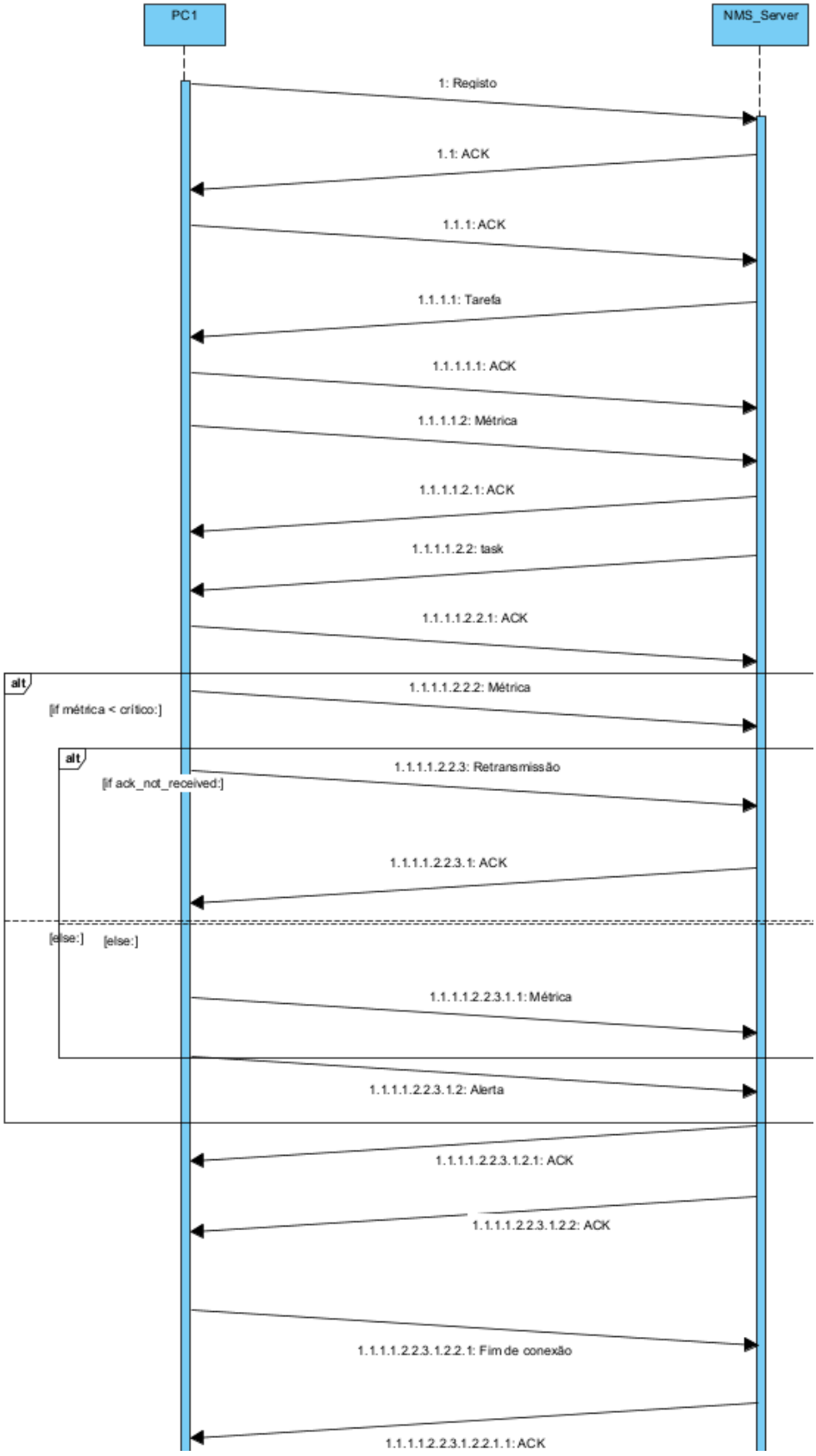



Figura 2: Diagrama de sequência de troca de mensagens entre o NMS_Server e 2 agentes

# 4. Implementação

A linguagem utilizada neste projeto é o python devido à sua sintaxe simples que facilita o desenvolvimento do código, bem como a clareza que proporciona.

## 4.1. Detalhes, parâmetros e bibliotecas de funções

Python oferece diversas bibliotecas bem suportadas que são diretamente aplicáveis aos requisitos do projeto como é o caso da biblioteca **socket** que foi utilizada para suportar a comunicação UDP e TCP de forma a criar e gerir os sockets utilizados para ambos os protocolos.

Foi também utilizada a biblioteca **psutil** para tratar da monitorização de recursos do sistema, como o CPU e a RAM.

Outra biblioteca usada para tratar da monitorização de métricas é a **subprocess** que é útil para executar ferramentas externas, tal como o ping e iperf, necessárias para medir latência, jitter e bandwidth.

Por fim usamos a biblioteca **struct** que permite uma manipulação eficiente de dados binários para criar mensagens protocolares compactas.

## 4.2. Implementação do NMS_Agent

O agente regista-se no servidor enviando uma mensagem de registo (tipo 1) ao servidor, recebendo uma resposta de confirmação (ACK) e enviando um ACK de forma ao servidor saber que este recebeu a sua confirmação, sendo que após isto, o agente espera por tarefas.

Com base no tipo de tarefa, executa medições da métrica pretendida, codificando-os e envia-as ao servidor, esperando por um ACK antes de passar para a próxima medição. Caso o agente não receba um ACK, este tenta retransmitir a medição feita, desistindo após 3 tentativas. Se a métrica capturada estiver acima das condições críticas, envia alertas através do TCP.

## 4.3. Implementação do NMS_Server

O servidor recebe as mensagens de registo e confirma o recebimento destas, e após receber a confirmação de receção vinda do agente, regista o agente na tabela de agentes, identificados por ID e endereço IP, registados no servidor.

Através da leitura de um ficheiro JSON, o servidor envia as respetivas tarefas aos agentes registados e usa mensagens binárias para enviar os parâmetros e receber os resultados das métricas medidas pelos agentes.

A todo o momento o servidor está atento a qualquer alerta que seja enviado pelo agente, sendo que estes, quando são recebidos, são mostrados ao utilizador.

## 4.4. Implementação dos Sockets e Resiliência

Os socket para o UDP e TCP foram criados através da biblioteca **socket**, onde o socket UDP é usado para a comunicação usada para tarefas e resultados, enquanto que o TCP é usado para alertas.

Foram definidos timeouts para evitar bloqueios no recebimento das mensagens, sendo implementadas, por norma, 3 tentativas de retransmissão para garantir a entrega.

Usamos threads paralelas de modo a permitir que o servidor lide com múltiplas conexões simultâneas, tarefas e alertas.

# 5. Teste e resultados

Nesta secção residem os testes que realizamos bem como os seus resultados.

## 5.1. Testes Funcionais

Ao longo do desenvolvimento do programa fomos confirmando que as métricas foram enviadas e recebidas corretamente, porém, devido a dificuldades técnicas, nem sempre foi possível testar todas métricas, tendo sido maior parte do tempo testadas as métricas referentes ao sistema (CPU e RAM), tendo que as outras métricas não usufruiram de tanto refinamento.

O envio de alertas foi testado juntamente do envio de métricas, sendo este um sucesso, visto que sempre que a métrica ultrapassa o threshold, o agente envia sempre o alerta correspondente.

## 5.2. Resultados

Após várias medições, o consumo médio do CPU é de 3% enquanto estão 2 agentes a enviar métricas e cerca de 40% no consumo de RAM. A latência média medida está por volta dos 12ms quando medida do “PC1” para o “PC3”, estando o servidor colocado no “PC2”. Enquanto que o packet loss, jitter e bandwidth estão a 0%, 1ms e 105 MBit/s.

# 6. Conclusões e trabalho futuro

## 6.1. Conclusões

O projeto foi bem-sucedido em criar um sistema modular e eficiente para monitorização de métricas em ambientes distribuídos. Maior parte dos objetivos principais foram atingidos, incluindo a comunicação confiável entre servidor e agentes e a deteção de condições críticas, ficando uma melhor retransmissão como principal falha.

## 6.2. Trabalho Futuro

Apesar de já termos uma boa base, existe, claramente, como melhorar, sendo que o que melhoraríamos num trabalho futuro seria a retransmissão das mensagens, que neste momento não está tão bem implementada; um mecanismo que faça um tipo de "queue" das tarefas por agente, não sendo necessário que o utilizador envie as tarefas manualmente; e também a implementação de um sistema baseado em DNS para a substituição de IPs estáticos.