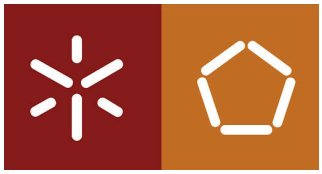

5 de dezembro de 2024

Relatório do Trabalho Prático 2 de Comunicações por Computadores

Rafael Lopes Seara, a104094

Pedro Manuel Dias Teixeira, a103998

Ricardo Gomes de Sousa, a104524

Conteúdo

1. Introdução

Este relatório tem como objetivo principal apresentar o trabalho prático desenvolvido para a unidade curricular de Comunicações por Computador. O trabalho consiste no desenvolvimento de um **Sistema de Gestão de Redes (*Network Monitoring System* - NMS)**, baseado num modelo **cliente-servidor distribuído**. Este sistema tem como finalidade monitorizar continuamente o estado dos dispositivos e dos *links* numa rede, recolhendo métricas relevantes e gerando alertas em caso de anomalias.

O projeto incluiu a implementação de dois protocolos aplicacionais: ***NetTask*** (utilizando UDP) para a comunicação de tarefas e recolha de métricas, e ***AlertFlow*** (utilizando TCP) para a notificação de eventos críticos.

Ao longo deste relatório, serão apresentadas a arquitetura do sistema, a especificação dos protocolos desenvolvidos, a implementação, bem como as decisões tomadas durante o desenvolvimento deste projeto.

Uma vez que a escolha da linguagem de programação era livre, optámos por ***Python*** devido à familiaridade da equipa com a linguagem e à sua simplicidade, que facilitou a implementação dos conceitos abordados.

2. Arquitetura da Solução

Desde o início do desenvolvimento, a arquitetura foi planeada para cumprir todos os requisitos solicitados, garantindo uma separação clara de responsabilidades e eficiência na monitorização de redes. O sistema é composto por um **servidor centralizado (*NMS Server*)** e múltiplos **agentes distribuídos (*NMS Agents*)**, seguindo o modelo **cliente-servidor**. Esta abordagem permite uma comunicação robusta e escalável entre os componentes, com o servidor a desempenhar o papel de coordenador principal e os agentes a recolherem e enviarem métricas e alertas.

O servidor utiliza dois protocolos aplicacionais, ***NetTask* (usando UDP)** e ***AlertFlow* (usando TCP)**, para gerir diferentes fluxos de comunicação. O protocolo *NetTask* é responsável pela troca de tarefas e métricas de desempenho entre o servidor e os agentes, enquanto o protocolo *AlertFlow* lida com a notificação de eventos críticos na rede. Cada agente executa comandos locais para recolher informações da rede e envia os dados ao servidor. A **interface do utilizador (UI)** permite ao gestor aceder aos dados armazenados, analisar métricas e visualizar alertas, garantindo a eficiência na gestão da rede. O diagrama ilustra a organização modular do sistema.

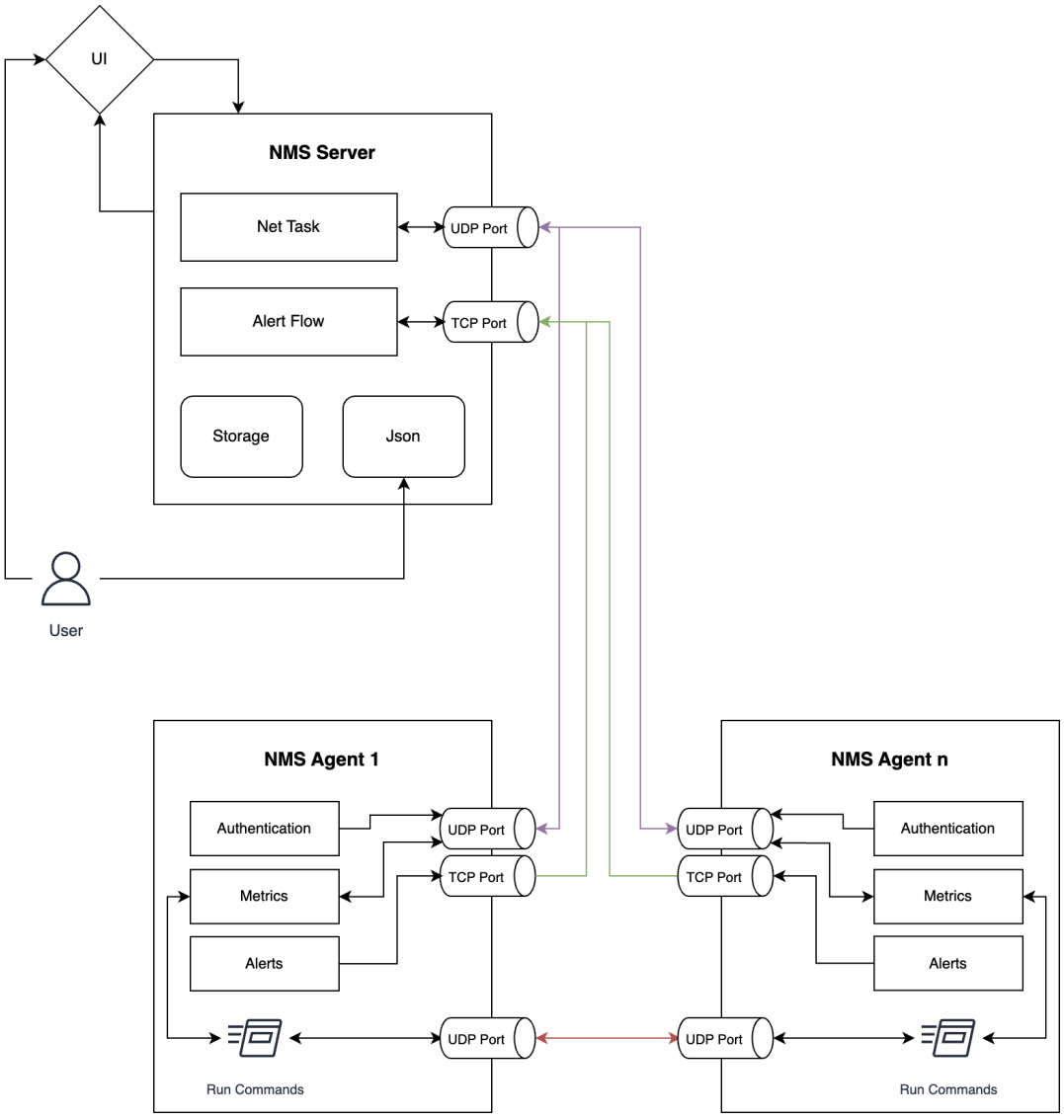

Figura 1: Arquitetura Geral

3. Especificação dos Protocolos

Os dois protocolos implementados no projeto, ***NetTask*** e ***AlertFlow***, desempenham papéis distintos, mas complementares, na comunicação entre o servidor central (*NMS Server*) e os agentes distribuídos (*NMS Agents*). Ambos os protocolos utilizam sockets para troca de mensagens, sendo o *NetTask* baseado no protocolo UDP e o *AlertFlow* baseado no protocolo TCP.

3.1. Formato das Mensagens

3.1.1. *NetTask*

O protocolo *NetTask* é utilizado para a comunicação de tarefas do servidor para os agentes e para a recolha contínua de métricas dos agentes. Ele foi desenvolvido para ser leve e eficiente, aproveitando a simplicidade do UDP.

As mensagens enviadas e recebidas no *NetTask* utilizam o formato **JSON** para facilitar a interpretação. Exemplos de mensagens incluem:

Envio de Tarefas (Servidor para Agentes):

```
{
  "task_id": "task-202",
  "frequency": 20,
  "device_id": "1",
  "device_metrics": {
    "cpu_usage": true,
    "ram_usage": true,
    "interface_stats": ["eth0", "eth1"]
  },
  "link_metrics": {
    "bandwidth": {
      "enabled": true,
      "tool": "iperf",
      "role": "server",
      "server_address": "127.0.0.1",
      "duration": 10,
      "transport_type": "TCP",
      "frequency": 30
    },
    "latency": {
      "enabled": true,
      "tool": "ping",
      "destination": "127.0.0.1",
      "packet_count": 5,
      "frequency": 20
    }
  },
  "alertflow_conditions": {
    "cpu_usage": 80.0,
    "ram_usage": 90.0
  }
}
```

Recolha de Métricas:

```
{
        "cpu_usage": 42.9,
        "ram_usage": 85.5,
        "link_metrics": {
            "bandwidth": {
                "bandwidth": "0/894 (0%)",
                "jitter": "0.022 ms",
                "packet_loss": "0/894 (0%)"
            },
            "latency": null
        }
    },
```

Fluxo de Comunicação:

1. O agente registra-se no sevidor.
2. O servidor envia tarefas periodicamente para os agentes.
3. Os agentes recolhem métricas e enviam-nas de volta para o servidor.
4. Caso o servidor não receba um **ACK** após o envio de uma mensagem, retransmite-a com um limite de tentativas configurado.

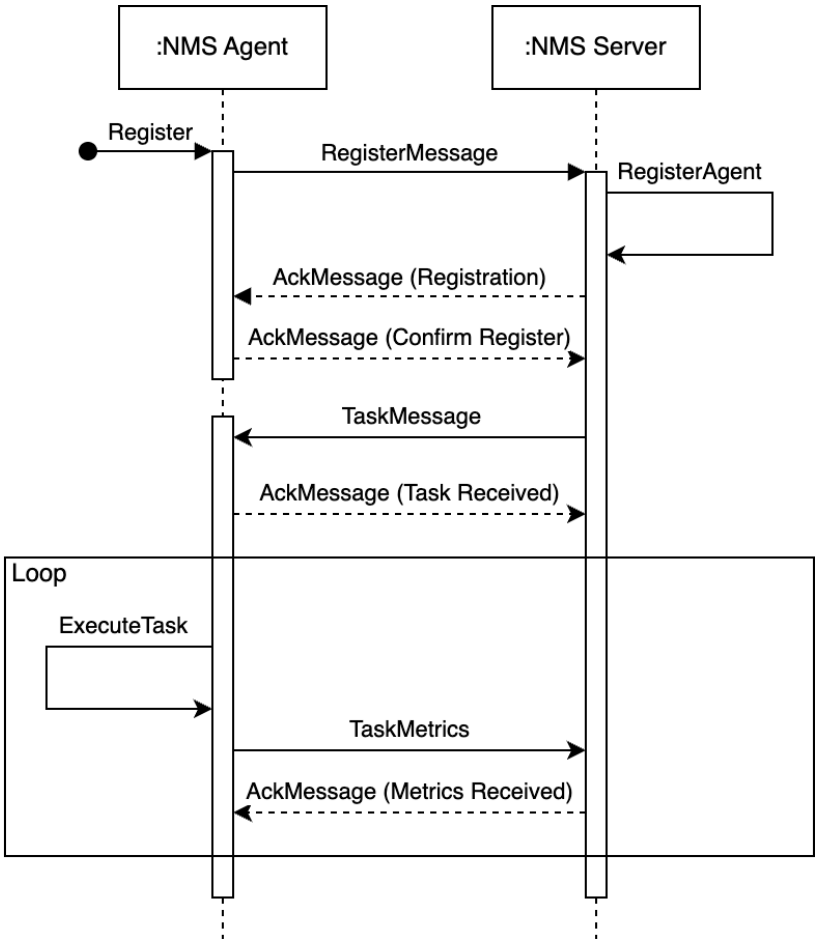

Figura 2: Diagrama de Sequência *NetTask* enre agente e servidor

Mecanismos de Resiliência:

O protocolo implementa retransmissões manuais, simulando confiabilidade sobre UDP. A função `send_with_retransmission` reenvia mensagens até que um **ACK** seja recebido ou o número de tentativas se esgote.

3.1.2. *AlertFlow*

O protocolo *AlertFlow* é usado para notificar eventos críticos e alterações no estado da rede. Devido à importância e sensibilidade dessas notificações, ele utiliza o TCP para garantir a entrega confiável das mensagens.

Assim como o NetTask, o AlertFlow utiliza mensagens em formato JSON. Exemplos incluem:

Notificação de Alerta:

```
{
    "alert": "High RAM Usage",
    "value": 92.5,
    "agent_id": "2",
    "threshold": 85.0
}
```

ACK do Servidor:

```
{
    "status": "ack",
    "alert_received": true
}
```

Fluxo de Comunicação:

1. O agente deteta uma condição crítica (ex.: utilização de CPU acima de um limiar).
2. Envia uma notificação de alerta para o servidor através de uma conexão TCP.
3. O servidor processa o alerta e envia um ACK de confirmação.
4. Após o envio, a conexão TCP é encerrada.

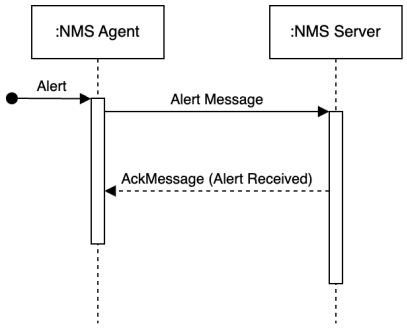

Figura 3: Diagrama de Sequência *AlertFlow* enre agente e servidor

Mecanismos de Confiabilidade

Graças ao TCP, a entrega das mensagens é garantida, e a retransmissão é gerida automaticamente pelo protocolo. Além disso, o servidor envia um ACK para confirmar a receção e processamento do alerta.

4. Implementação

4.1. *NMS Server*

4.1.1. *Parsing* do JSON

O servidor utiliza uma função específica (`load_task_config`) para interpretar o ficheiro de configuração em formato JSON. Este ficheiro contém os detalhes das tarefas, incluindo:

- Identificação da tarefa (`task_id`).
- Frequência da recolha de métricas.
- Configuração dos dispositivos, como métricas a recolher e condições de alerta.

O JSON é carregado através da biblioteca padrão `json` em *Python*, sendo transformado numa estrutura de dados em memória. Essa estrutura é usada para atribuir tarefas aos agentes e validar os parâmetros enviados.

4.1.2. Gestão de Tarefas

Após carregar o JSON, o servidor gere as tarefas com os seguintes passos:

Atribuição de Tarefas:

- As tarefas configuradas são enviadas para os agentes registados através do protocolo ***NetTask***.
- O envio é feito na função `send_task_to_agents`, onde cada agente recebe apenas os dados relevantes ao seu `device_id`.

Recolha de Métricas:

- O servidor aguarda mensagens dos agentes com os dados recolhidos.
- As métricas recebidas são armazenadas em ficheiros JSON individuais para cada agente (armazenados na pasta `metrics_storage`).

Retransmissão (Resiliência):

- O protocolo ***NetTask*** implementa uma camada de retransmissão para lidar com potenciais perdas de pacotes no UDP.

4.1.3. Interface para o Gestor

A interface de utilizador (UI) foi implementada utilizando a biblioteca curses em *Python* e oferece funcionalidades como:

- a **visualização de *logs***, permitindo ao gestor consultar os registos de comunicação e operação;
- a **consulta de métricas**, com um menu que possibilita selecionar agentes e visualizar os dados recolhidos;
- a **gestão de configuração**, que permite carregar um ficheiro JSON de configuração diretamente pela interface.

4.2. *NMS Agent*

4.2.1. Coleta de Métricas

Os agentes utilizam ferramentas externas para recolher métricas de desempenho. Cada agente segue as configurações fornecidas pelo servidor para recolher métricas com a frequência especificada no JSON. Os resultados são formatados em JSON e enviados para o *NMS Server* através do protocolo ***NetTask***

4.2.2. Comunicação com Outros Agentes

No caso de ferramentas como o `iperf`, a comunicação entre agentes é estabelecida para medir métricas como largura de banda. Isso é configurado no JSON do servidor, onde:

- Um agente pode ser configurado como cliente (`role: client`) ou servidor (`role: server`).
- O endereço do servidor (`server_address`) é partilhado, garantindo que os testes de largura de banda são coordenados.

5. Testes e Resultados

5.1. Cenário de Teste

Para validar o funcionamento do sistema, os testes foram realizados utilizando a topologia `CC-Topo-2024.imn` configurada no simulador CORE. Esta topologia emula uma rede com múltiplos nós distribuídos, representando os ***NMS Agents***, e um servidor central, o ***NMS Server***. Os testes incluíram:

- Distribuição de *NMS Agents*: Posicionamento dos agentes em diferentes localizações na topologia para verificar a comunicação em cenários variados.
- Condições de Rede Adversas: Introdução de latência artificial, perda de pacotes e variação de largura de banda para simular redes reais.

Para reproduzir os testes na topologia `CC-Topo-2024.imn` no simulador CORE, foi necessário realizar os seguintes passos:

Instalar as dependências necessárias para o servidor:

```
sudo apt install ncurses-term
export TERM=xterm
python3 Server-Side/NMS_Server.py
```

Executar os agentes nos nós designados na topologia:

```
python3 Agent-Side/NMS_Agent.py
```

5.2. Resultados

Os testes realizados no simulador CORE validaram a implementação do ***Network Monitoring System (NMS)***, demonstrando a capacidade do sistema em gerir agentes distribuídos, recolher métricas detalhadas e lidar com condições adversas de rede.

5.2.1. Execução no CORE

Para a execução no simulador CORE os agentes foram posicionados em diferentes nós da rede. Durante os testes, foi possível verificar:

- O registro bem-sucedido dos agentes no servidor.
- A comunicação contínua e eficiente entre agentes e servidor, mesmo em condições de rede com latência elevada e perdas de pacotes simuladas.
- A possibilidade de carregar um ficheiro de configuração JSON.

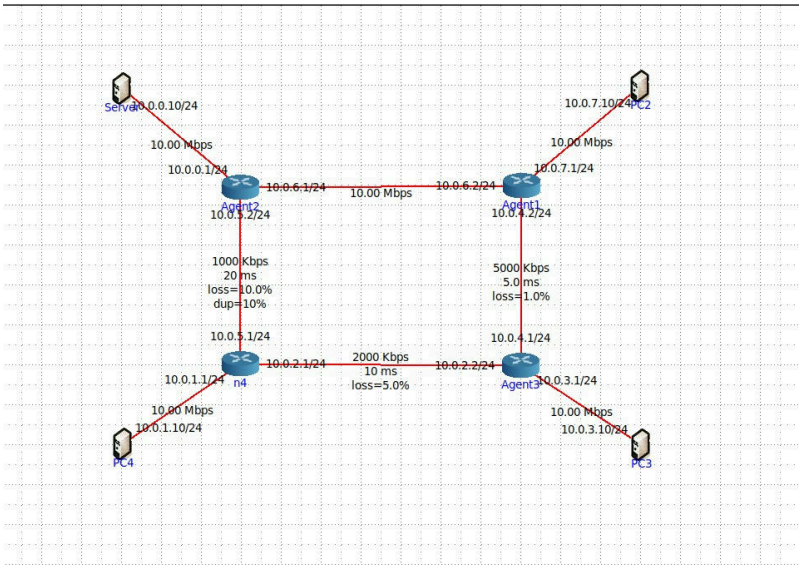

Figura 4: Topologia configurada no CORE

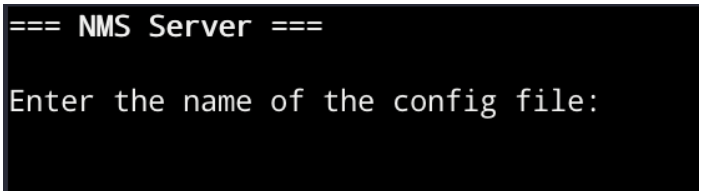

Figura 5: Input do ficheiro JSON para configuração

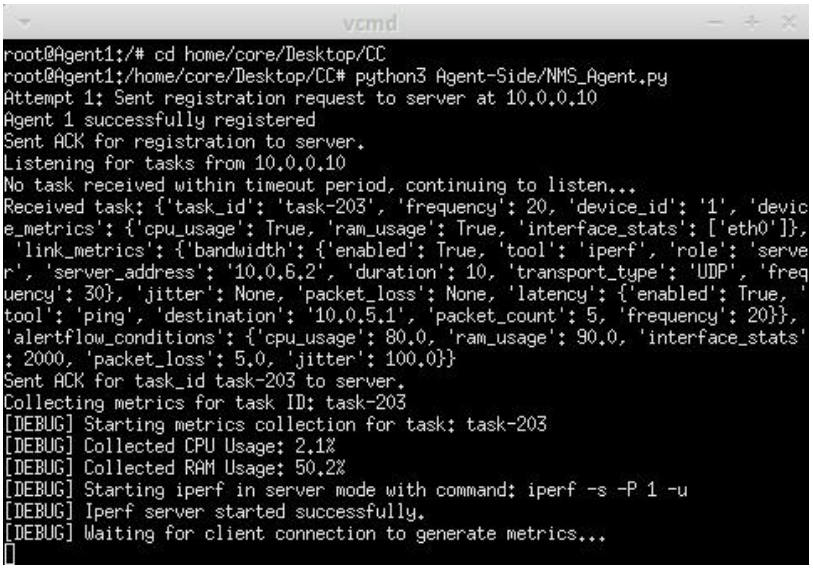

Figura 6: Configuração Agente 1

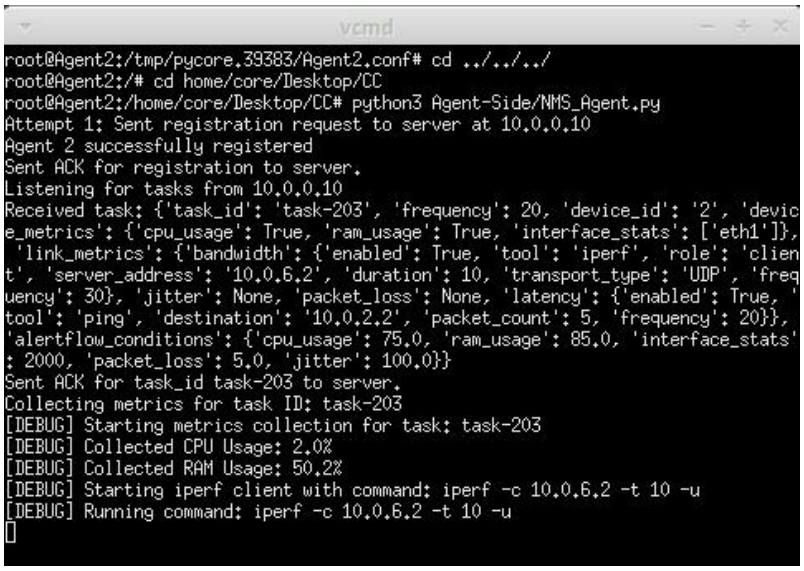

Figura 7: Configuração Agente 2

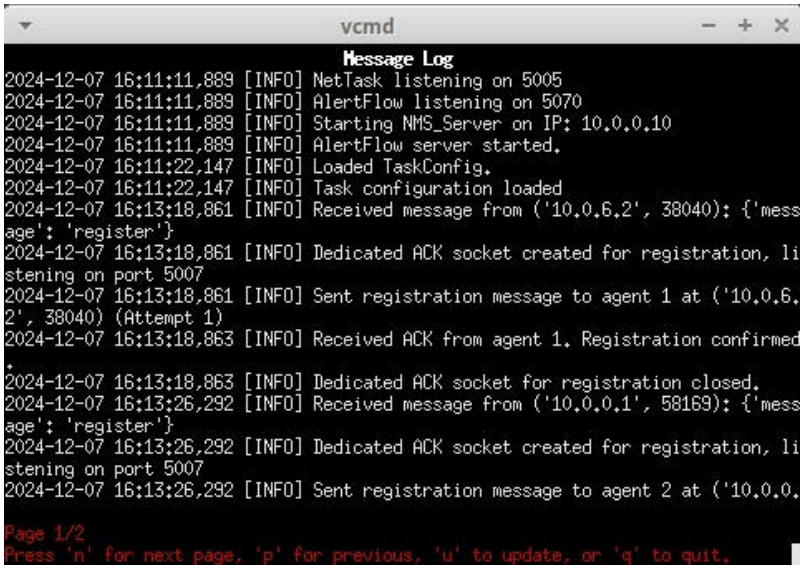

Figura 8: Mensagens de configuração do Servidor (Interface - Message Log)

5.2.2. Registo e Coleta de Métricas

O NMS Server mostrou-se capaz de registar todos os agentes conectados e monitorizar métricas em tempo real. Foram registados logs detalhados no servidor, demonstrando a receção de mensagens de registo, tarefas distribuídas e métricas recolhidas.

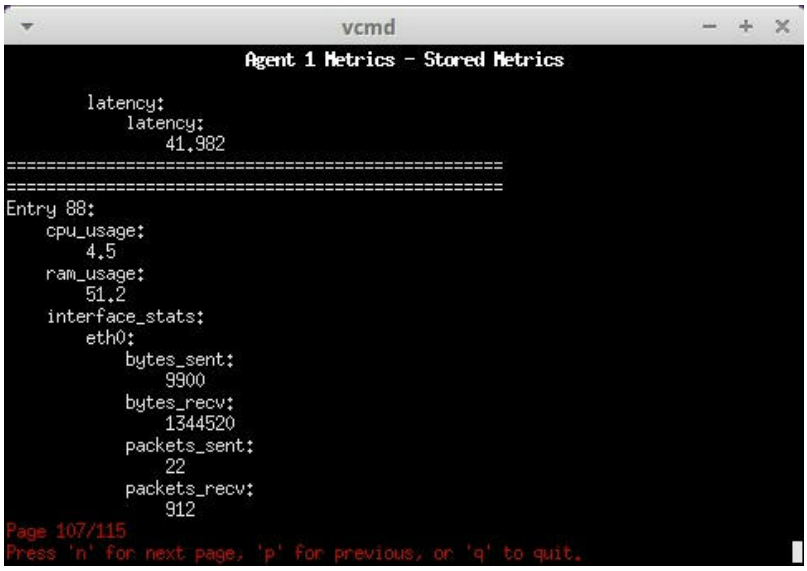

Figura 9: Interface - Stored Metrics (Agent 1)

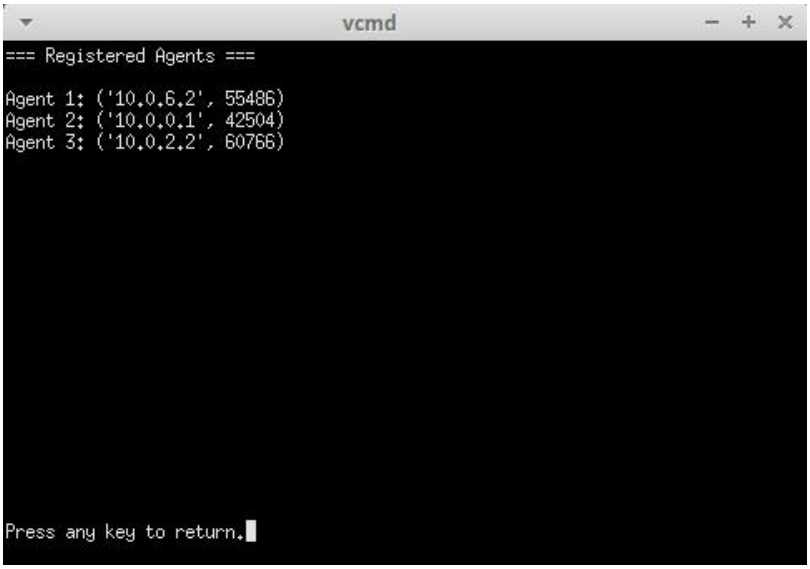

Figura 10: Interface - Registerd Agents

Figura 11: Interface - Message Log da receção de métricas

Estes resultados confirmam a funcionalidade e resiliência do sistema desenvolvido, demonstrando que este cumpre os objetivos propostos para monitorizar redes e gerar alertas em tempo útil.

6. Conclusão e Trabalho Futuro

O desenvolvimento deste projeto resultou na criação de um Sistema de Gestão de Redes (*Network Monitoring System - NMS*) eficiente e funcional, capaz de monitorizar continuamente dispositivos e links numa rede. Através da implementação de protocolos aplicacionais (*NetTask* e *AlertFlow*), foi possível estabelecer uma comunicação robusta e fiável entre o servidor central e os agentes distribuídos, assegurando a recolha de métricas e a deteção de anomalias em tempo real.

A arquitetura modular, combinada com uma interface de utilizador intuitiva, permitiu centralizar a visualização e gestão de métricas, contribuindo para uma gestão de rede mais eficiente. O sistema demonstrou resiliência em cenários adversos e adaptabilidade a diferentes condições de rede. No entanto, há potencial para expandir as funcionalidades, como a inclusão de monitorização mais avançada e suporte a redes mais complexas, reforçando a utilidade do sistema no contexto de redes modernas. Este trabalho não só consolidou os conhecimentos adquiridos durante a unidade curricular, como também reforçou a importância de soluções distribuídas e escaláveis para a gestão de redes.