Relatório TP2 | Comunicações por Computador Grupo 1.3 | 2024/2025

João Lobo[A104356], Mariana Rocha[A90817], and Rita Camacho[A104439]

Universidade do Minho, Braga, Portugal
a104356@uminho.pt and a90817@uminho.pt and a104439@uminho.pt
https://www.uminho.pt/PT

1 Introdução

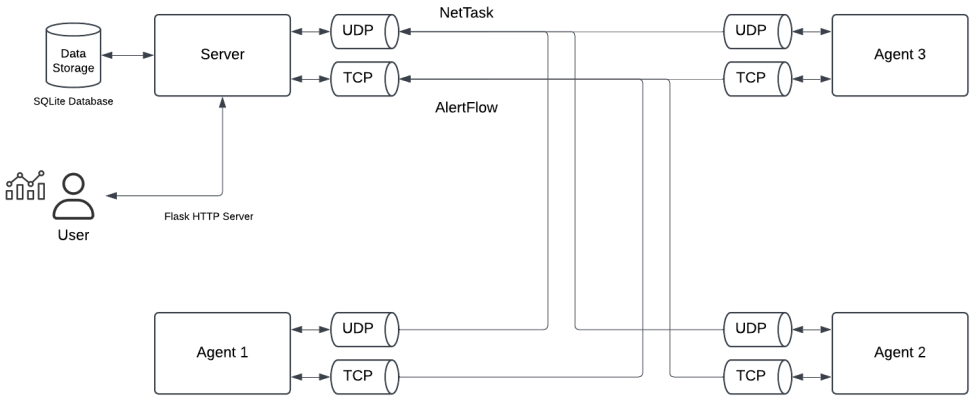
O presente relatório apresenta informações relativas ao trabalho realizado como 2ª Parte da Componente Prática da Unidade Curricular Comunicações por Computador, pertencente ao 3º Ano da Licenciatura em Engenharia Informática, realizada no ano letivo 2024/2025, na Universidade do Minho.

Este trabalho consiste no desenvolvimento de um sistema (distribuído) de gestão de redes capaz de fornecer informação detalhada sobre o estado dos *links* e dos dispositivos numa rede, bem como gerar alertas caso sejam detetadas anomalias. Este sistema tem como base um modelo cliente-servidor onde uma aplicação cliente é responsável por recolher diferentes métricas de interesse e de as reportar a um servidor centralizado.

Para o desenvolvimento deste projeto foi utilizada a linguagem de programação Python, tendo esta sido escolhida pelo grupo.

2 Arquitetura da solução

De forma a cumprirmos todas as funcionalidades requisitadas pela equipa docente, adotamos a seguinte arquitetura na realização do projeto:



(a) Arquitetura do projeto

2.1 Componentes

Servidor

Atua como o núcleo do sistema, responsável pelo envio de tarefas para coleta de métricas de rede da topologia aos agentes. Conecta-se a um sistema de base de dados SQLite para armazenar informações persistentes, como métricas e alertas.

Agentes
Estes estão hospedados em dispositivos distribuídos na rede, conectados ao servidor via UDP e TCP, utilizam ferramentas de coleta de métricas de rede no sistema para enviar resultados ao servidor.

Utilizador
Interage com o sistema através de uma *interface web* baseada em `Flask` fornecida pelo servidor, permitindo consulta das métricas recolhidas.

2.2 Fluxos de comunicação

NetTask
Protocolo aplicacional que usa a camada de transporte UDP e incorpora características normalmente associadas ao TCP. Responsável pela comunicação de tarefas e resultados da coleta contínua de métricas.

AlertFlow
Protocolo aplicacional que usa a camada de transporte TCP. Responsável pela comunicação de notificações de alterações críticas no estado dos dispositivos de rede.

3 Especificação dos protocolos propostos

3.1 Formato das mensagens protocolares

O formato das mensagens protocolares é uma parte essencial para garantir a comunicação eficiente e confiável entre os agentes e o servidor. Vamos descrever os formatos utilizados para cada tipo de pacote, detalhando os campos incluídos e sua disposição no *payload*. O sistema utiliza um protocolo baseado em UDP com características de confiabilidade e controlo, complementado por TCP para alertas.

Estrutura Geral das Mensagens do UDP

As mensagens seguem um esquema binário para eficiência e compactação, composto pelos seguintes elementos principais:

1. **Tipo do Pacote (1 byte):** Identifica o propósito da mensagem, como registo de agentes, envio de métricas ou controle de fluxo.
2. **Número de Sequência (1 byte):** Garante a ordenação e acompanhamento das mensagens.
3. **Número de ACK (1 byte):** Utilizado para confirmar o recebimento de mensagens, implementando confiabilidade.
4. **Payload Variável:** Contém os dados específicos da mensagem, dependendo do tipo.

Tipos de Mensagens e Formatos

0. Pacote do Registo do Agente (RegisterAgent)
Propósito: Registar um agente no servidor.

Campo	Tamanho (byte)	Descrição
Tipo do Pacote	1	0 para RegisterAgent
Número de Sequência	1	Sequência única da mensagem
Número de ACK	1	Sempre 0, pois não é resposta
ID do Agente	5	Identificador do agente

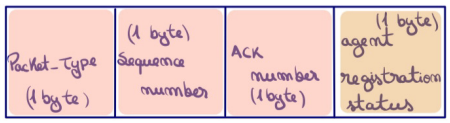
Table 1: RegisterAgent

(a) Formato do pacote RegisterAgent

1.Resposta do Registo (RegisterAgentResponse)
Propósito: Informar o *status* do registo de um agente.

Campo	Tamanho (byte)	Descrição
Tipo do Pacote	1	1 para RegisterAgentResponse
Número de Sequência	1	Sequência única da mensagem
Número de ACK	1	Sequência do pacote reconhecido
Status do Registo	1	Enum indicando Success, AlreadyRegistered, etc.

Table 2: RegisterAgentResponse

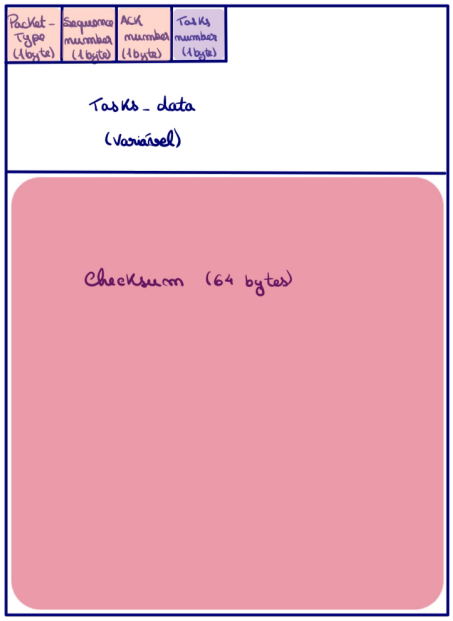


(a) Formato do pacote RegisterAgentResponse

2. Pacote da Tarefa (TaskPacket)
Propósito: Distribuir tarefas aos agentes.

Campo	Tamanho (byte)	Descrição
Tipo do Pacote	1	2 para TaskPacket
Número de Sequência	1	Sequência única da mensagem
Número de ACK	1	Sempre 0, pois não é resposta
Número de Tarefas	1	Quantidade de tarefas no pacote
Dados das Tarefas	Variável	Array serializado das tarefas
Checksum	64 bytes	SHA-256 para garantir integridade

Table 3: TaskPacket



(a) Formato do pacote TaskPacket

3. Pacote de Métricas (MetricsPacket)
Propósito: Enviar resultados das medições realizadas pelos agentes.

Campo	Tamanho (byte)	Descrição
Tipo do Pacote	1	3 para MetricsPacket
Número de Sequência	1	Sequência única da mensagem
Número de ACK	1	Sempre 0, pois não é resposta
ID da Tarefa	10	Identificador da tarefa
ID do Device	5	Identificador do agente
Largura de Banda	4	Medição em Mbps
Jitter	4	Jitter médio em ms
Perda de Pacotes	4	Percentagem de pacotes perdidos
Latência	4	Latência média em ms
Timestamp	4	Epoch time (segundos)
Checksum	64	SHA-256 para garantir integridade

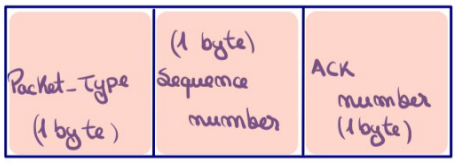
Table 4: MetricsPacket

(a) Formato do pacote MetricsPacket

4. Pacote de Confirmação (ACK)
Propósito: Confirmar a receção dos pacotes.

Campo	Tamanho (byte)	Descrição
Tipo do Pacote	1	4 para ACKPacket
Número de Sequência	1	Sequência única da mensagem
Número de ACK	1	Sequência do pacote reconhecido

Table 5: ACK



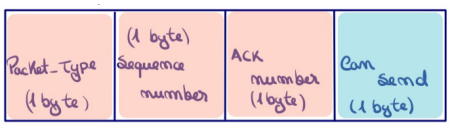
(a) Formato do pacote ACKPacket

5. Pacote de de Controlo de Fluxo (FlowControlPacket)

Propósito: Gerir o fluxo das mensagens, indicando se os agentes podem continuar a enviar pacotes.

Campo	Tamanho (byte)	Descrição
Tipo do Pacote	1	5 para FlowControlPacket
Número de Sequência	1	Sequência única da mensagem
Número de ACK	1	Sempre 0, pois não é resposta
Estado do Envio	5	1 (pode enviar) ou 0 (não enviar)

Table 6: FlowControlPacket



(a) Formato do pacote FlowControlPacket

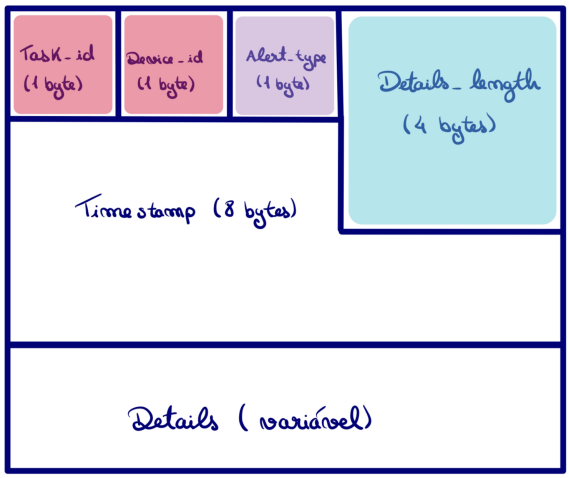
Estrutura Geral das Mensagens do TCP

Para a comunicação TCP, temos o pacote **AlertMessage** que desempenha um papel crucial dentro do sistema, sendo este responsável pelo envio dos alertas gerados pelos agentes para o servidor. Estes alertas são criados com base em condições críticas da tarefa já pré-definidas, como o uso elevado do CPU, perda de pacotes, ou valores altos do *jitter*. O pacote AlertMessage encapsula todas as informações necessárias para identificar e detalhar o alerta, garantindo que o servidor seja avisado através do TCP.

Os tipos de alertas possíveis são representados pelo **AlertType**. Com esta abordagem simplificámos a identificação de problemas críticos. O pacote AlertMessage é composto pelos seguintes campos:

Campo	Tamanho (byte)	Descrição
ID da Tarefa	1	Identificador da tarefa
ID do Device	1	Identificador do agente
Alert_Type	1	Tipo de alerta
Detalhes	Variável	Detalhes do alerta
Timestamp	8	Momento que o alerta foi gerado

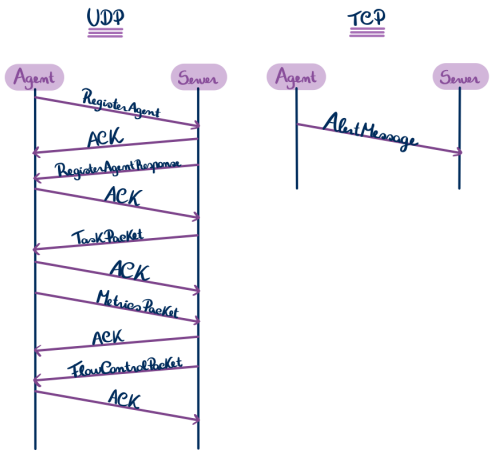
Table 7: RegisterAgent



(a) Formato do pacote AlertMessage

3.2 Diagrama de sequência da troca de mensagens

O diagrama apresentado descreve a troca de mensagens entre um Agente e um Servidor em dois cenários de comunicação, utilizando os protocolos UDP e TCP.



(a) Diagrama de Sequência da Troca de Mensagens

Na comunicação UDP, temos a necessidade de *ACKs*, sendo estes enviados após cada mensagem transmitida. O processo de comunicação começa com o Agente a enviar um *RegisterAgent* ao Servidor, que responde com um ACK a confirmar a receção. Em seguida, o Servidor responde com um *RegisterAgentResponse*, e o Agente confirma novamente com um *ACK*. A troca continua com o envio de pacotes de dados, como o *TaskPacket* e o *MetricsPacket*, com cada um a ser confirmado pelo respetivo *ACK*.

Na comunicação TCP, mostra que o Agente envia uma *AlertMessage* ao Servidor, sem a necessidade de *ACKs* adicionais, visto que o TCP gere as confirmações e a retransmissão das mensagens internamente.

O diagrama reflete um processo de comunicação robusto e eficiente, com a troca de pacotes de dados sendo esta realizada de forma confiável no caso do TCP, e no caso do UDP, onde a confiabilidade é garantida pelas confirmações de recebimento de cada pacote enviado.

4 Implementação

A implementação do sistema de comunicação no nosso projeto é baseada em um protocolo personalizado de troca de pacotes, utilizado para garantir uma comunicação eficiente e confiável entre os agentes e o servidor. O sistema foi desenvolvido para funcionar principalmente sobre o protocolo UDP, incorporando características de controlo de fluxo, retransmissão e garantias de entrega que são típicas do TCP. Isto é feito com o uso de números de sequência e pacotes de confirmação (ACKs) para garantir que as mensagens sejam entregues de forma ordenada e sem perdas.

4.1 Detalhes

O sistema de comunicação foi projetado para garantir uma interação eficiente entre agentes e o servidor, mesmo em condições adversas de rede. A abordagem implementada combina a leveza do protocolo UDP com mecanismos adicionais que trazem confiabilidade e controlo típicos de protocolos como o TCP. Esses mecanismos incluem:

- **Controlo de Fluxo**: Para evitar congestionamento, um sistema de fluxo bidirecional gere a permissão de envio de pacotes.
- **Garantia de Entrega**: Para assegurar que todas as mensagens são recebidas, são utilizados números de sequência em cada pacote enviado. No caso de falha num envio (transmissor não recebe ACK), a mensagem é retransmitida.
- **Ordenação de Mensagens**: A numeração sequencial dos pacotes permite que o sistema mantenha as mensagens na ordem correta, mesmo que os pacotes sejam recebidos fora de sequência devido às características do UDP.

4.2 Parâmetros e Estruturas de Dados

A comunicação entre os agentes e o servidor é realizada com o uso de diferentes tipos de pacotes. Cada pacote contém parâmetros que desempenham papéis críticos na operação do sistema. Assim, detalhamos os principais parâmetros utilizados nas mensagens e como eles influenciam o funcionamento do sistema:

1. Tipo do Pacote: Cada mensagem enviada entre o agente e o servidor é identificada por um tipo de pacote, que é especificado pelo campo *packet_ type*. Este campo define o propósito da mensagem e pode tomar os seguintes valores:

- RegisterAgent (0): Pacote utilizado para registar um novo agente no servidor
- RegisterAgentResponse (1): Resposta do servidor ao pacote de registo do agente, indicando o estado da operação (sucesso, agente já registado, etc.).
- Task (2): Pacote enviado para distribuir tarefas para os agentes.
- Metrics (3): Pacote enviado por um agente para reportar as métricas de desempenho.
- ACK (4): Pacote utilizado para confirmar a receção de pacotes e garantir a entrega ordenada.
- FlowControl (5): Pacote utilizado para gerir o fluxo de envio de mensagens entre o servidor e os agentes, controlando quando é que os pacotes podem ser enviados com base no estado da fila de cada agente ou servidor.

2. Número de Sequência: Cada envio de pacote contém um número de sequência exclusivo, registado no campo *sequence_ number*. Este número serve para identificar a sequência de envio dos pacotes e assegurar que as mensagens são tratadas na ordem adequada. Além disso, também auxilia na identificação de pacotes que são duplicados ou que foram perdidos.

3. Número de ACK: O campo *ack_ number* é usado para assegurar a fiabilidade na comunicação. Ao enviar um pacote, o recetor retorna um ACK com o número de sequência do pacote que foi corretamente recebido. Isso possibilita ao remetente ter conhecimento de que o pacote foi devidamente processado e que o seguinte pacote pode ser despachado. Caso o número de ACK não coincida com o número de sequência esperado, o pacote será reenviado.

4. Estado do Fluxo: No campo *can_ send*, que é usado no pacote de controlo de fluxo, um valor *True* ou *False*, que sinaliza se o servidor está a permitir o envio de novos pacotes. Esse controlo é essencial para prevenir a congestão, assegurando que o servidor recebe pacotes apenas quando estiver pronto para processá-los. Quando o fluxo é interrompido, os clientes esperam a indicação de que podem prosseguir com o envio de pacotes.

5. Dados Específicos de Métricas: No conjunto de métricas, atributos como largura de banda, *jitter*, perda de pacotes, latência e *timestamp* são utilizados para registar e reportar o desempenho da rede. Essas informações são captadas pelo agente e transmitidas ao servidor.

- Largura de Banda (*bandwidth*): Avaliada em Mbps, indicando a quantidade de informações que pode ser enviada por unidade de tempo.
- *Jitter* (*jitter*): Variação no tempo de chegada dos pacotes, frequentemente medida em milissegundos.
- Perda de Pacotes (*packet_ loss*): Proporção de pacotes que se perderam ao longo da transmissão.
- Latência (*latency*): O tempo de caminho de um pacote de dados, calculado em milissegundos.
- *Timestamp* (*timestamp*): A indicação temporal do instante em que a métrica foi capturada.

6. Alertas (*Alert Type*): A troca de alertas entre o servidor e os agentes ocorre por meio de pacotes do tipo *AlertMessage*. Esses pacotes incluem dados sobre a categoria do alerta em questão e as informações sobre a situação crítica identificada.

6.1. *AlertType*: Enumeração que especifica os tipos de alertas disponíveis:

- *HIGH_ JITTER*: Detetado excesso de jitter.
- *HIGH_ PACKET_ LOSS*: Perda de pacotes superior ao limite aceitável.
- *HIGH_ CPU_ USAGE*: Consumo excessivo do CPU no agente.
- *HIGH_ RAM_ USAGE*: Consumo excessivo da memória RAM no agente.
- *HIGH_ INTERFACE_ STATS*: Estatísticas de interface acima do limite.

7. Pacotes para Controlo do Fluxo: A gestão de fluxo é realizada com a utilização do pacote *FlowControlPacket*. O servidor pode enviar pacotes com *can_ send* configurado como *False* para indicar que o agente não deve enviar mais pacotes até que o servidor esteja pronto a recebê-los. Quando o fluxo é autorizado, o servidor envia um pacote com *can_ send* marcado como *True*, possibilitando que o agente prossiga com o envio de pacotes.

4.3 Biblioteca de funções

O projeto envolve a comunicação entre o servidor e agentes, e para implementar essa comunicação de forma eficaz, foram criadas diversas bibliotecas personalizadas. Essas bibliotecas abrangem desde o controlo de fluxo até à gestão de métricas de rede e envio de alertas. De seguida, estão descritas as funcionalidades e o propósito de algumas das bibliotecas desenvolvidas.

`server.py`

A biblioteca `server.py` foi desenvolvida para centralizar todos os aspetos relacionados com o servidor, desde registo de agentes, como distribuição de tarefas pelos mesmos. É também realizado o registo e processamento de métricas e alertas, enviados pelos agentes.

Função Principal: Registo de agentes, distribuição de tarefas, registo e processamento de métricas e alertas.

`agent.py`

A biblioteca `agent.py` foi desenvolvida para centralizar todos os aspetos relacionados com os agentes desde o seu registo, como o cálculo e envio de métricas, e realização de alertas.

Função Principal: Cálculo de métricas, realização de alertas.

`server/agents_manager.py`

A biblioteca `agents_manager.py` foi desenvolvida para permitir o registo de agentes, assim como o `get` dos mesmos (na totalidade ou individualmente, sendo o último obtido pelo ID).

Função Principal: Gerir os agentes.

Exemplos de uso: Registar os agentes.

`server/database.py`

A biblioteca `database.py` foi desenvolvida para dar *setup* e permitir interação com uma base de dados, recorrendo a SQLite. Foram criadas duas tabelas, `packets` e `alertflow`.

Função Principal: Armazenar informações dos pacotes e alertas.

Exemplos de uso: Interface recorre à base de dados para demonstrar a informação.

`server/tasks_json.py`

A biblioteca `tasks_json.py` foi desenvolvida para importar as tarefas a distribuir posteriormente pelos agentes registados.

Função Principal: Importar ficheiro .JSON com as tarefas.

Exemplos de uso: Depois da sua importação, permite distribuir as tarefas pelos agentes.

`agent/conditions.py`

A biblioteca `conditions.py` foi desenvolvida para calcular estatísticas, como o uso de RAM e CPU.

Função Principal: Calcular estatísticas importantes.

Exemplos de uso: Estatísticas reveladas na interface.

`agent/metrics.py`

A biblioteca `metrics.py` foi desenvolvida para calcular métricas, como a latência e perda de pacotes.

Função Principal: Calcular métricas importantes.

Exemplos de uso: Métricas reveladas na interface.

`agent/tools.py`

A biblioteca `tools.py` foi desenvolvida para permitir que o programa utilize primitivas de sistema como o `ping` e `iperf` e recolha / interprete os seus resultados.

Função Principal: Realização de `ping` e `iperf`.

Exemplos de uso: Realização de `ping` e `iperf`.

`web/app.py`

A biblioteca `app.py` foi desenvolvida, recorrendo aos *templates* (inseridos em `web/templates`), para o projeto ter uma interface `web`, de forma a que visualizar pacotes, métricas, gráficos de alertas, etc. fosse possível, de forma mais interativa.

Função Principal: Criar uma interface.

Exemplos de uso: Visualizar gráficos de alertas.

`lib/logging.py`

A biblioteca `logging.py` foi desenvolvida para gerir a geração e registo de *logs* no terminal, ou seja, esta permite que o sistema registe mensagens com diferentes níveis de informação (informação, erro, debug) e *timestamps*, o que facilita a análise e monitorização da aplicação.

Função Principal: Fornecer um sistema simples de *log*, permitindo o acompanhamento do fluxo de dados e identificação de problemas durante a execução do sistema.

Exemplos de uso: Regista eventos importantes como a criação de conexões, envio de pacotes, erros na comunicação, entre outros.

`lib/packets.py`

A biblioteca `packets.py` define a estrutura e o processamento dos pacotes enviados entre o servidor e os agentes. Os pacotes são divididos em diferentes tipos, incluindo pacotes de registo, métricas, ACKs e controlo de fluxo. A biblioteca também é responsável pela serialização e desserialização dos dados para enviar e receber pacotes no formato correto.

Função Principal: Gerir a criação, serialização e verificação de pacotes de dados trocados entre o servidor e os agentes. Esta define os tipos de pacotes, condições de controlo e a estrutura de cada tipo de pacote.

Exemplos de uso: Criação de pacotes para registar um agente no servidor.

`lib/tcp.py`

A biblioteca `tcp.py` lida com a comunicação entre o servidor e os agentes com o protocolo de comunicação TCP. Esta permite que os agentes enviem alertas para o servidor, garantindo que as mensagens são entregues de forma confiável. Utiliza a abordagem de pacotes de alerta para informar o servidor de eventos críticos.

Função Principal: Facilitar o envio de alertas com o TCP entre o servidor e os agentes, garante a entregas das mensagens críticas.

Exemplos de uso: Os agentes enviam pacotes de alerta sobre condições críticas pré-definidas.

`lib/udp.py`

A biblioteca `udp.py` implementa a comunicação entre o servidor e os agentes com o protocolo de comunicação UDP. Esta biblioteca tem características extra na comunicação: controlo de fluxo, retransmissão e confirmação de entrega dos pacotes.

Função Principal: Gerir a receção e o envio de pacotes usando UDP. Também implementa o controlo de fluxo entre o servidor e os agentes, garantindo que os pacotes sejam enviados apenas quando o servidor estiver pronto para os receber.

Exemplos de uso: Receber pacotes de métricas ou enviar tarefas para um agente.

`lib/task.py e lib/task_serializer.py`

As bibliotecas `task.py` e `task_serializer.py` são responsáveis pela estruturação das tarefas que o servidor envia para os agentes. Estas definem as características das tarefas, incluindo o conjunto de métricas a serem recolhidas e os *devices* envolvidos.

Função Principal: Criar e serializar/deserializar tarefas para os agentes. Isso envolve a conversão de dados de tarefas para um formato que pode ser facilmente transmitido entre o servidor e os agentes.

Exemplos de uso: As tarefas são serializadas para serem enviadas como pacotes de rede e podem ser desserializadas pelos agentes para a sua execução.

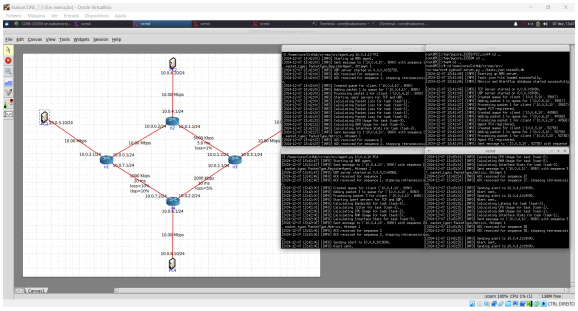
4.4 DNS

A implementação de DNS foi também acrescentada, segundo as orientações transmitidas em aula prática.

Esta foi realizada configurando o `Name Server` do `Linux Bind` no **CORE** (`named.conf.local`), alterando os seus ficheiros de configuração e criando uma zona para o projeto (`/etc/bind/zones/cc2024.zone`) - nesta, ligou-se identificadores de dispositivos, como **PC1**, a endereços IP.

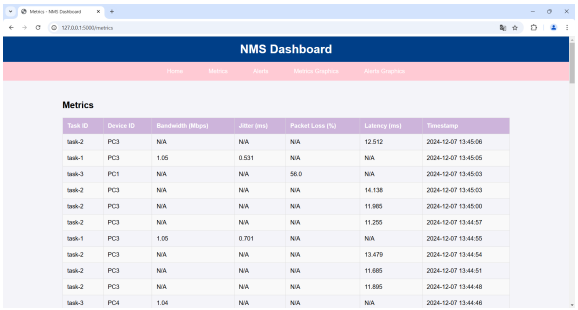
5 Testes e resultados

Para validar o nosso projeto, realizámos diferentes testes na seguinte topologia:



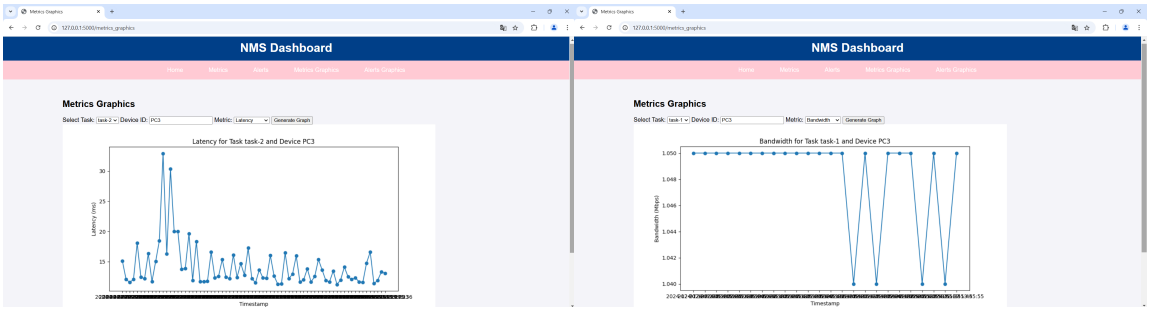
(a) Topologia

Os agentes foram configurados para analisar o tráfego em tempo real, onde foram registadas as métricas pedidas - largura de banda, jitter, perdas de pacotes e latência - diretamente na base de dados SQLite. Estas métricas podem ser acedidas na aba **Metrics** da aplicação *web*, onde são apresentadas de forma tabular.



(a) Métricas

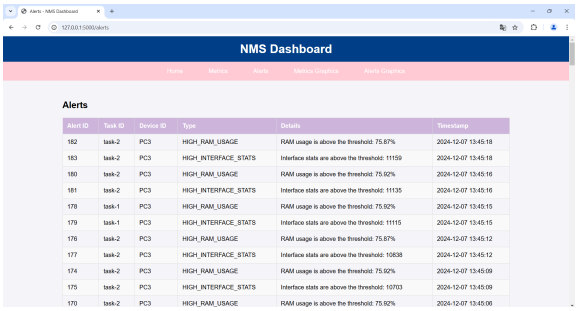
Os gráficos disponíveis na aba **Metrics Graphics** permitem a análise temporal destas métricas, conseguindo identificar potenciais problemas em momentos específicos. Por exemplo, observámos picos de latência até 30 ms no dispositivo PC3, claramente representados no gráfico.



(a) Gráfico de Métricas - Latency (b) Gráfico de Métricas - Bandwith

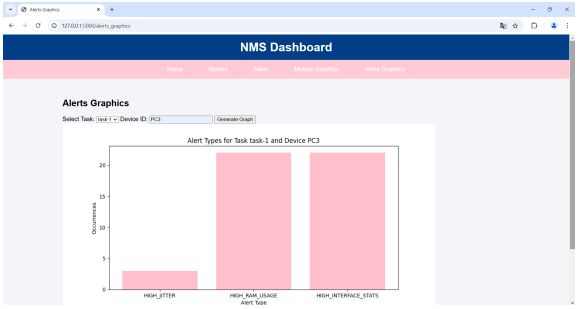
Fig. 12: Gráficos de métricas

Além da recolha de métricas, testámos a geração de alertas para cenários específicos. Estes alertas foram também registados na base de dados SQLite e podem ser vistos na aba **Alerts** com diversos detalhes - tipo de alerta, tarefa, dispositivo associado, uma descrição e o timestamp correspondente.



(a) Alertas

Na aba **Alerts Graphics**, temos gráficos adicionais que mostram a frequência que ocorreu cada tipo de alerta, o que facilitou a identificação de padrões de comportamento diferente do habitual.



(a) Gráfico de Alertas

Tendo em conta os diferentes testes realizados, concluímos que o sistema tem uma recolha e armazenamento eficaz das métricas, bem como conseguimos apresentar uma visualização das métricas para termos essa perceção. Os gráficos ajudam a analisar as tendências e anomalias que possam eventualmente acontecer, os alertas também ajudaram a detetar rapidamente os eventos críticos. Estes resultados validam a solução em cenários reais, onde temos uma análise de métricas e deteção de problemas continuamente, sendo isto essencial para uma gestão eficiente.

6 Conclusões

Concluindo, acreditamos ter alcançado o pretendido neste trabalho. Todas as dificuldades e dúvidas foram esclarecidas graças ao apoio do professor nas aulas práticas, algo que nos permitiu alcançar mais facilmente os nossos objetivos. A definição de tarefas semanais também se revelou uma ótima orientação para organização interna do grupo e, claro, do próprio trabalho final. Este projeto, apesar de desafiante, melhorou os nossos conhecimentos lecionados na unidade curricular de uma forma prática.