

# Trabalho Prático Nº2 – Monitorização Distribuída de Redes

Carolina Queirós<sup>1[a100836]</sup> and Tomás Rodrigues<sup>1[a104448]</sup>

<sup>1</sup> Universidade do Minho, Braga, Portugal

**Abstract.** O sistema implementado monitoriza redes em uma arquitetura cliente-servidor, com os módulos NMS\_Agent e NMS\_Server. O NMS\_Agent coleta métricas de latência, largura de banda e uso de CPU usando as bibliotecas ping3, subprocess e psutil. Essas métricas são enviadas ao NMS\_Server via UDP (NetTask), enquanto alertas de valores críticos são transmitidos via TCP (AlertFlow). O servidor armazena métricas e alertas em arquivos JSON e ajusta a frequência de envio com base no controle de fluxo.

**Keywords:** UDP, TCP, Python.

## 1 Introdução

O presente trabalho prático tem como objetivo implementar um sistema de monitorização distribuída de redes utilizando conceitos de redes de computadores e protocolos de comunicação. Este sistema é composto por dois módulos principais: o NMS\_Server, responsável por centralizar e processar as métricas e alertas recebidos, e o NMS\_Agent, que executa tarefas de monitorização em dispositivos remotos e envia informações ao servidor. A implementação baseia-se nos protocolos NetTask (UDP) e AlertFlow (TCP), os quais foram especificados para garantir o envio confiável de métricas e a notificação de alertas. As métricas recolhidas incluem parâmetros como latência, largura de banda, e outros indicadores de desempenho da rede, essenciais para a avaliação do estado e funcionamento dos sistemas monitorizados. Este projeto foi desenvolvido com base na topologia do TP1, conforme especificado no enunciado, decidimos utilizar a linguagem de programação Python e bibliotecas relevantes. Os testes foram realizados no emulador CORE, de forma a validar o correto funcionamento das implementações.

## 2 Arquitetura da Solução

A arquitetura deste projeto é baseada no modelo cliente-servidor, no qual o NMS\_Agent coleta métricas de rede, em particular, latência, uso de CPU e largura de banda, e envia-as, através do protocolo NetTask, para o NMS\_Server. O protocolo AlertFlow é ativado apenas quando as métricas excedem um determinado limite.

De modo a testar o funcionamento destes protocolos foi desenvolvido uma topologia que permite simular diversas qualidades de rede.

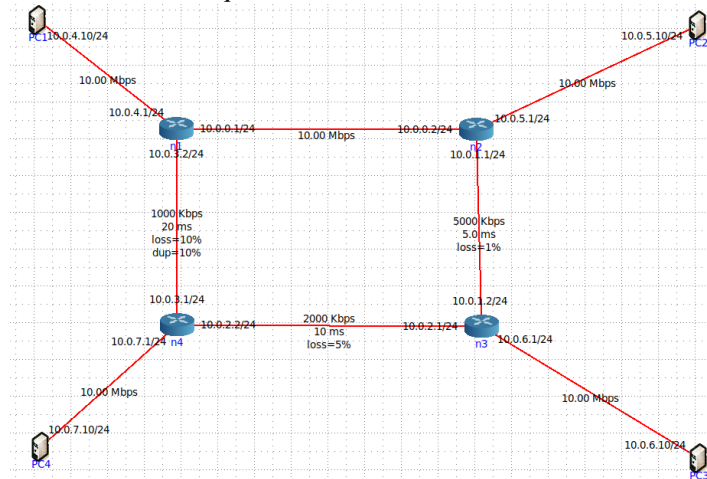


Figura 1 - Topologia na qual serão testados os protocolos desenvolvidos

A topologia utilizada é composta por 4 hosts, cada host está conectado a um único router, e 4 routers. Os routers estão conectados através de 4 ligações com diferentes valores de débito, atraso, perdas e duplicação, sendo a conexão  $n1 \rightarrow n2$  a mais confiável e  $n1 \rightarrow n4$  a menos. As ligações entre os hosts e os routers têm as mesmas características, garantindo que os resultados serão provenientes das diferenças na qualidade das conexões entre os routers.

Ao executar os protocolos desenvolvidos na topologia, o NMS\_Server deve ser executado num dos 4 PCs e, posteriormente, o NMS\_Agent deve ser executado nos routers indicando qual o ip do servidor com o qual este deve interagir.

### 3 Especificação dos Protocolos NetTask (UDP) e AlertFlow (TCP)

#### 3.1 Formato das Mensagens Protocolares

##### Protocolo NetTask (UDP)

Header:

- msg\_type (1 byte): tipo da mensagem (1 para registro de agente, 2 para envio de métricas, 3 para ACK)
- sequence\_num (2 bytes): número de sequência para garantir a ordem das mensagens
- agent\_id (2 bytes): identificador único do NMS\_Agent
- checksum (2 bytes): para garantir a integridade dos dados

Payload (corpo da mensagem):

- task\_id (2 bytes): id da tarefa associada à métrica
- metric\_type (1 byte): tipo da métrica
- metric\_value (4 bytes): valor da métrica coletada
- timestamp (8 bytes): marca temporal do dado coletado

### Protocolo AlertFlow (TCP)

Header:

- agent\_id (4 bytes): identificador único do NMS\_Agent.

Payload:

- alert\_type (1 byte): tipo de alerta
- metric\_type (1 byte): tipo da métrica relacionada ao alerta (latência, perda de pacotes, etc.)
- metric\_value (4 bytes): valor da métrica que gerou o alerta
- threshold (4 bytes): limite que foi excedido para gerar o alerta
- timestamp (8 bytes): marca temporal do alerta

### Mecanismos de Controlo

#### NetTask:

- Número de Sequência => cada mensagem terá um número de sequência incrementado a cada envio; permite garantir a ordem e conhecer os pacotes que não foram recebidos;
- Checksum => checksum pode ser utilizado para verificar a integridade dos pacotes

#### AlertFlow:

Uma vez que o Protocolo AlertFlow usa o TCP, não é necessário implementar mecanismos de controlo ao nível da camada de aplicação, pois o próprio TCP implementa retransmissão e controlo de fluxo na camada de transporte.

## 3.2 Diagrama de Sequência Data-Troca de Mensagens

- **Registo (NetTask):**
  - Agente → Servidor: Regista o NMS\_Agent.
  - Servidor → Agente: Confirmação de registo (ACK).
- **Solicitação de Tarefas (NetTask):**
  - Servidor → Servidor: Envia uma tarefa para coletar métricas (definindo quais métricas e com que frequência).
  - Agente → Agente: Confirmação de recebimento da tarefa (ACK).
- **Envio de Métricas (NetTask):**
  - Agente → Servidor: Envio periódico de métricas.
  - Servidor → Agente: ACK para cada métrica recebida.
- **Alerta Crítico (Alert Flow):**
  - Agente → Servidor: Notificação de alerta.

- Servidor → Agente: Confirmação do alerta (ACK).



Figura 2 - Diagrama de Sequência Data-Troca de Cada Tipo de Mensagem

## 4 Implementação

### 4.1 Detalhes da Execução

Ao executar o NMS\_Agent é necessário passar os parâmetros `config_file` e `agent_id` para identificar o agente e o servidor com o qual este deve interagir (`config.json` → `localhost` ou `configPCx.json` → `PCx`, sendo `x` o valor que identifica o PC da simulação ao qual este se refere).

Em seguida, o NMS\_Agent irá interpretar o `config_file` e configurar o agente através da função `load_agent_config`. Após a sua configuração, o agente vai enviar para o servidor um pedido de registo através da função `register_agent` (`msg_type=1`). Caso o registo seja bem sucedido, o servidor responde com um `ACK` (`msg_type=3`) e o NMS\_Agent começa a processar tarefas. Cada tarefa é executada em uma thread separada, permitindo que múltiplas métricas sejam coletadas simultaneamente. A função `send_metric` coleta a métrica correspondente ao `task_id` (Latência: 1, Uso de CPU: 2, Largura de Banda: 3) e envia-a (`msg_type=2`). Quando o valor de uma métrica

ultrapassar o threshold definido no config\_file, o NMS\_Agent envia um alerta ao NMS\_Server via TCP. As métricas e alertas enviados ao servidor serão guardadas nos ficheiros metrics.config e alerts.config.

## 4.2 Parâmetros

A troca de dados entre o NMS\_Agent e o NMS\_Server ocorre através das funções pack unpack da biblioteca struct.

### Funções no NMS\_Agent

- register\_agent(agent\_id, server\_ip, udp\_port)
  - message=struct.pack('!BHHH', msg\_type, sequence\_num, agent\_id, checksum)
  - ack\_msg\_type, ack\_sequence\_num, ack\_agent\_id, ack\_checksum=struct.unpack('!BHHH', data)
- send\_metric(task, agent\_id, server\_ip, udp\_port)
  - message = struct.pack('!BHHHIBIQH', 2, sequence\_num, agent\_id, checksum, task\_id, metric\_type, metric\_value, timestamp, 0)
  - ack\_msg\_type, ack\_sequence\_num, ack\_agent\_id, ack\_checksum, flow\_control\_flag = struct.unpack('!BHHHB', data)
- send\_alert(agent\_id, metric\_type, metric\_value, threshold, server\_ip, tcp\_port)
  - struct.pack('!HBBIIQ', agent\_id, 1, metric\_type, metric\_value, threshold, int(time.time()))

### Funções no NMS\_Server

- listen\_udp()
  - msg\_type, sequence\_num, agent\_id, checksum = struct.unpack('!BHHH', data)
  - \_, sequence\_num, agent\_id, checksum, task\_id, metric\_type, metric\_value, timestamp, \_ = struct.unpack('!BHHHIBIQH', data)
- listen\_tcp()
  - agent\_id, alert\_type, metric\_type, metric\_value, threshold, timestamp = struct.unpack('!HBBIIQ', data)

O empacotamento e desempacotamento destes dados é feito segundo a formatação indicada:

- ! - ordem de bytes (big endian)
- B - unsigned char (1 byte)
- H - unsigned short (2 bytes)
- Q - unsigned long long (8 bytes)

Deste modo, garantem-se as restrições anteriormente definidas na formatação para os protocolos NetTask e AlertFlow.

### 4.3 Bibliotecas de Funções

O código desenvolvido em python faz uso de 5 bibliotecas, **socket**, **struct**, **json**, **time**, **ping3**, **psutil** e **subprocess**.

A biblioteca **socket** é essencial para a comunicação entre o **NMS\_Agent** e o **NMS\_Server**, permitindo o uso de UDP para transmissão de métricas e TCP para o envio de alertas. O empacotamento das diversas componentes dos dados trocados foi realizado através da biblioteca **struct**. A biblioteca **json** foi necessária para ler e guardar dados em ficheiros de formato json, como **config**, **metrics** e **alerts**. O timestamp dos protocolos é obtido através da biblioteca **time**. Por último, recorreu-se às bibliotecas **ping3**, **psutil** e **subprocess** para calcular as métricas de Latência, Uso de CPU e Largura de Banda, respetivamente.

## 5 Testes e Resultados

Depois de tudo implementado decidimos testar o **NMS\_Server** e o **NMS\_Agent** primeiro no windows e depois no emulador CORE. Para isso utilizamos a topologia e iniciamos o servidor num dos PCs e o Agente num ou mais routers.

```

root@PC1:/tmp/picore-36717/PC1.conf# cd /home/core/Desktop/CC-tp2
root@PC1:/tmp/picore-36717/PC1.conf# python3 NMS_Server.py
Servidor: iperf3 iniciado.
NMS_Server emulando UDP na porta 5000
NMS_Server emulando TCP na porta 5000
Recebido 7 bytes: b'\x01\x00\x00\x00\x02\x00\x00\x00'
Registro recebido do agente com ID: 2
Recebido 28 bytes: b'\x02\x00\x00\x00\x00\x02\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
Métrica recebida - Agente: 2, Tipo: 3, Valor: 12, Timestamp: 1733565578
Recebido 7 bytes: b'\x01\x00\x00\x00\x00\x02\x00\x00\x00'
Registro recebido do agente com ID: 1
Recebido 28 bytes: b'\x02\x00\x00\x00\x00\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
Métrica recebida - Agente: 1, Tipo: 1, Valor: 44, Timestamp: 1733565581
Código TCP estabelecido com ('10.0.1.1', 5438)
Alerta recebido do Agente 1 - Tipo: 1, Valor: 44, Limite: 43, Timestamp: 1733565581
Alerta salvo: {'agent_id': 1, 'metric_type': 1, 'metric_value': 44, 'threshold': 43, 'timestamp': 1733565581}
Recebido 28 bytes: b'\x02\x00\x00\x00\x00\x02\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
Métrica recebida - Agente: 2, Tipo: 3, Valor: 12, Timestamp: 1733565589
Recebido 28 bytes: b'\x02\x00\x00\x00\x00\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
Métrica recebida - Agente: 1, Tipo: 1, Valor: 42, Timestamp: 1733565594
Recebido 28 bytes: b'\x02\x00\x00\x00\x00\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
Métrica recebida - Agente: 1, Tipo: 1, Valor: 152, Timestamp: 1733565616
Alerta recebido do Agente 1 - Tipo: 1, Valor: 152, Limite: 43, Timestamp: 1733565616
Alerta salvo: {'agent_id': 1, 'metric_type': 1, 'metric_value': 152, 'threshold': 43, 'timestamp': 1733565616}
Recebido 28 bytes: b'\x02\x00\x00\x00\x00\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
Métrica recebida - Agente: 2, Tipo: 3, Valor: 12, Timestamp: 1733565619
Recebido 28 bytes: b'\x02\x00\x00\x00\x00\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
Métrica recebida - Agente: 1, Tipo: 1, Valor: 43, Timestamp: 1733565626
Recebido 28 bytes: b'\x02\x00\x00\x00\x00\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
Métrica recebida - Agente: 2, Tipo: 3, Valor: 12, Timestamp: 1733565640
Recebido 28 bytes: b'\x02\x00\x00\x00\x00\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
Métrica recebida - Agente: 1, Tipo: 1, Valor: 43, Timestamp: 1733565640
Métrica recebida - Agente: 1, Tipo: 1, Valor: 43, Timestamp: 1733565640

```

```

root@n1:/tmp/picore-36717/n1.conf# cd /home/core/Desktop/CC-tp2
root@n1:/tmp/picore-36717/n1.conf# python3 NMS_Agent.py configPC1.json 2
Tentando registrar o agente com ID: 2 no servidor...
Registro confirmado para o agente com ID: 2
Métrica 'bandwidth' enviada e confirmada.
Métrica 'bandwidth' enviada e confirmada.
Métrica 'bandwidth' enviada e confirmada.
Métrica 'bandwidth' enviada e confirmada.

```

```

root@n4:/tmp/picore-36717/n4.conf# cd /home/core/Desktop/CC-tp2
root@n4:/tmp/picore-36717/n4.conf# python3 NMS_Agent.py configPC1.json 1
Tentando registrar o agente com ID: 1 no servidor...
Registro confirmado para o agente com ID: 1
Métrica 'latency' enviada e confirmada.
Preparando alerta: Agente 1, Métrica 1, Valor 44, Limite 43
Alerta enviado: Agente 1, Métrica 1, Valor 44, Limite 43
Métrica 'latency' enviada e confirmada.
Erro ao calcular métrica 'latency'. Ignorando envio.
Métrica 'latency' enviada e confirmada.
Preparando alerta: Agente 1, Métrica 1, Valor 152, Limite 43
Alerta enviado: Agente 1, Métrica 1, Valor 152, Limite 43
Métrica 'latency' enviada e confirmada.
Erro ao calcular métrica 'latency'. Ignorando envio.
Métrica 'latency' enviada e confirmada.

```

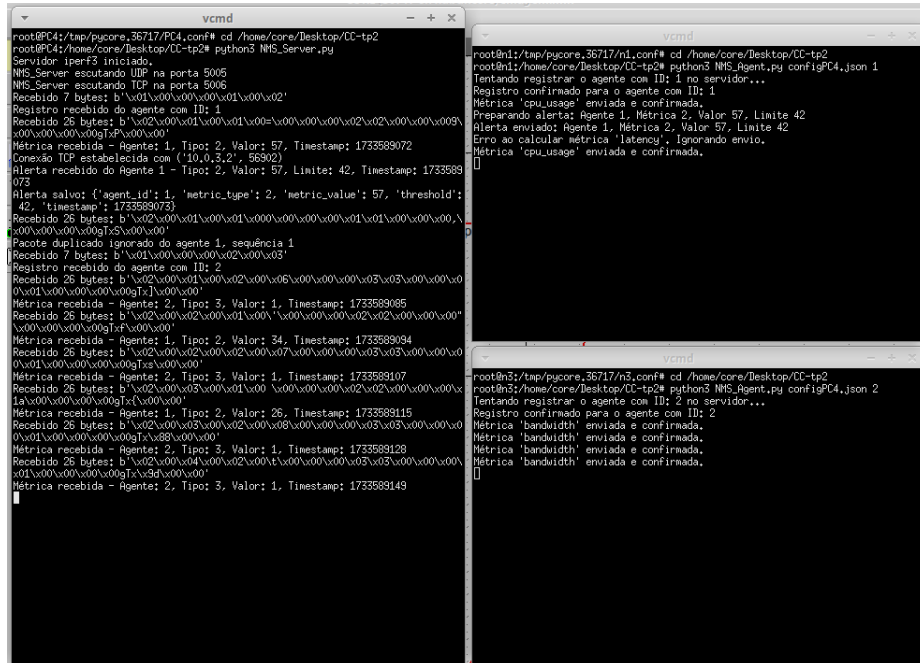
Figura 3 - Execução do servidor no PC1 e de Agentes nos router n1 e n4

Para um primeiro teste decidimos executar o servidor no PC1 com o comando **python3 NMS\_Server.py** e iniciar o agente em dois routers distintos com o comando **python3 NMS\_Agent.py configPC1.json 1** para executar a tarefa 1 e **python3 NMS\_Agent.py configPC1.json 2** para executar a tarefa 2. Como pudemos verificar para a tarefa 1 foi medida a “latência” e se o seu valor for inferior ao “threshold” é enviado para o servidor e se for superior é enviado um alerta. Neste teste conseguimos verificar os dois casos. Na tarefa 2 todos os valores forem inferiores logo todas as métricas “bandwidth” foram enviadas e confirmadas.









```

root@PC4:/tmp/pycore,36717/PC4,conf# cd /home/core/Desktop/CC-tp2
root@PC4:/home/core/Desktop/CC-tp2# python3 NMS_Server.py
Servidor Iperf3 iniciado.
NMS_Server escutando UDP na porta 5005
Recebido 7 bytes: b'\x01\x00\x00\x00\x01\x00\x02'
Registro recebido do agente com ID: 1
Recebido 26 bytes: b'\x02\x00\x00\x01\x00\x01\x00=\x00\x00\x02\x00\x00\x009\x00\x00\x00\x00gTx\x00\x00'
Métrica recebida - Agente: 1, Tipo: 2, Valor: 57, Timestamp: 1733589072
Conexão TCP estabelecida com ('10.0.3.2', 56902)
Alerta recebido do Agente 1 - Tipo: 2, Valor: 57, Limite: 42, Timestamp: 1733589073
Alerta salvo: {'agent_id': 1, 'metric_type': 2, 'metric_value': 57, 'threshold': 42, 'timestamp': 1733589073}
Recebido 26 bytes: b'\x02\x00\x00\x01\x00\x01\x00=\x00\x00\x01\x01\x00\x00\x00\x00\x00\x00gTx\x00\x00'
Pacote duplicado ignorado do agente 1, sequência 1
Recebido 7 bytes: b'\x01\x00\x00\x00\x02\x00\x03'
Registro recebido do agente com ID: 2
Recebido 26 bytes: b'\x02\x00\x01\x00\x02\x00\x06\x00\x00\x00\x03\x03\x00\x00\x00\x01\x00\x00\x00gTx\x00\x00'
Métrica recebida - Agente: 2, Tipo: 3, Valor: 1, Timestamp: 1733589085
Recebido 26 bytes: b'\x02\x00\x02\x00\x02\x00\x07\x00\x00\x00\x02\x02\x00\x00\x00\x00\x00gTx\x00\x00'
Métrica recebida - Agente: 1, Tipo: 2, Valor: 34, Timestamp: 1733589094
Recebido 26 bytes: b'\x02\x00\x02\x00\x02\x00\x07\x00\x00\x00\x03\x03\x00\x00\x00\x01\x00\x00\x00gTx\x00\x00'
Métrica recebida - Agente: 2, Tipo: 3, Valor: 1, Timestamp: 1733589107
Recebido 26 bytes: b'\x02\x00\x03\x00\x01\x00\x00\x00\x00\x02\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00gTx\x00\x00'
Métrica recebida - Agente: 1, Tipo: 2, Valor: 26, Timestamp: 1733589115
Recebido 26 bytes: b'\x02\x00\x03\x00\x02\x00\x08\x00\x00\x00\x03\x03\x00\x00\x00\x01\x00\x00\x00gTx\x00\x00'
Métrica recebida - Agente: 2, Tipo: 3, Valor: 1, Timestamp: 1733589128
Recebido 26 bytes: b'\x02\x00\x04\x00\x02\x00\x01\x00\x00\x00\x03\x03\x00\x00\x00\x01\x00\x00\x00gTx\x00\x00'
Métrica recebida - Agente: 2, Tipo: 3, Valor: 1, Timestamp: 1733589149

root@n1:/tmp/pycore,36717/n1,conf# cd /home/core/Desktop/CC-tp2
root@n1:/home/core/Desktop/CC-tp2# python3 NMS_Agent.py configPC4.json 1
Tentando registrar o agente com ID: 1 no servidor...
Registro confirmado para o agente com ID: 1
Métrica 'cpu_usage' enviada e confirmada.
Preparando alerta: Agente 1, Métrica 2, Valor 57, Limite 42
Alerta enviado: Agente 1, Métrica 2, Valor 57, Limite 42
Erro ao calcular métrica 'latency', ignorando envio.
Métrica 'cpu_usage' enviada e confirmada.

root@n3:/tmp/pycore,36717/n3,conf# cd /home/core/Desktop/CC-tp2
root@n3:/home/core/Desktop/CC-tp2# python3 NMS_Agent.py configPC4.json 2
Tentando registrar o agente com ID: 2 no servidor...
Registro confirmado para o agente com ID: 2
Métrica 'bandwidth' enviada e confirmada.
Métrica 'bandwidth' enviada e confirmada.
Métrica 'bandwidth' enviada e confirmada.

```

Figura 6 - Execução do servidor no PC4 e de Agentes nos router n1 e n3

No teste do PC4 também podemos verificar tudo o que foi verificado nos testes anteriores.

## 6 Conclusões e Trabalho Futuro

A implementação deste sistema neste trabalho permitiu-nos explorar e aplicar conceitos fundamentais relacionados com comunicação em redes e monitorização distribuída. Através da integração do **NMS\_Server** e do **NMS\_Agent**, foi possível realizar a recolha e análise de métricas essenciais, como latência, largura de banda e utilização de CPU, garantindo a monitorização contínua do estado da rede. Os testes realizados validaram a robustez e a funcionalidade do sistema, demonstrando a capacidade de detetar métricas fora dos limites predefinidos e de gerar alertas de forma eficaz. A utilização de protocolos específicos, como o **NetTask** e o **AlertFlow**, foi essencial para a fiabilidade da comunicação, destacando a importância de mecanismos de controlo de fluxo e integridade de pacotes. No entanto, durante o desenvolvimento, foram identificados desafios relacionados com a integração de ferramentas externas, como o **Iperf3** e o **Ping3** mas com pesquisa e ajuda nas aulas foram ultrapassados. Apesar dos resultados satisfatórios alcançados, pensamos que existem diversas possibilidades para melhorar e expandir o sistema desenvolvido tais como correção de alguns erros que podem existir no código e que não conseguimos resolver, também podemos vir a expandir o sistema de modo a incluir outras métricas diferentes que não foram utilizadas. Com estas

melhorias, achamos que o nosso sistema poderá ser utilizado em cenários reais, oferecendo uma solução completa e flexível para monitorização de redes.