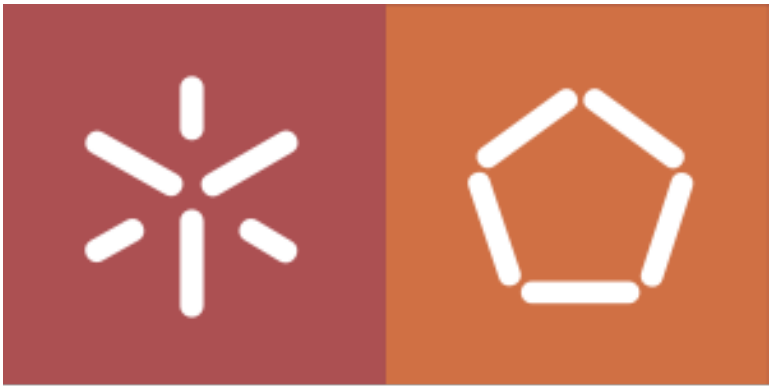


Universidade do Minho
Escola de Engenharia

Comunicações por Computador 24/25 – PL4

Trabalho Prático Nº2 – Monitorização Distribuída de Redes

Trabalho realizado por:

Diogo Silva a104183 | Diogo Silva a97941 | João Pinto a104270

Grupo 7 - TP2

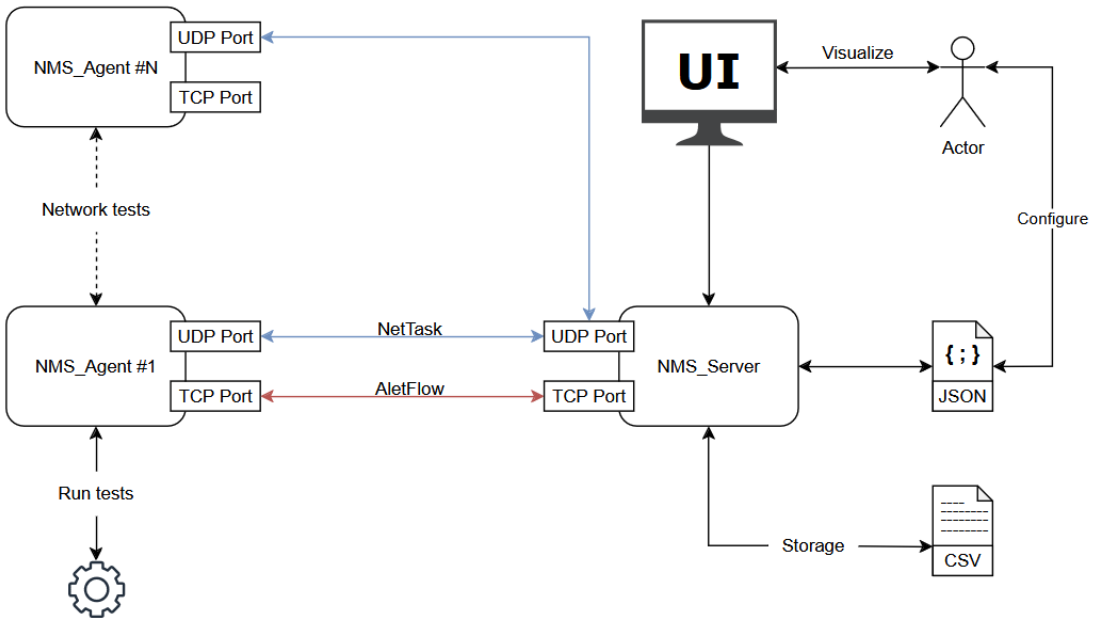
Introdução

A importância do monitoramento de rede está na sua capacidade de identificar e solucionar problemas antes que estes afetem os usuários finais, de modo a assegurar uma operação interrupta e eficiente na comunicação entre sistemas em ambientes modernos.

Este trabalho tem como objetivo implementar um Sistema de Monitorização de Redes (*Network Monitoring System* - NMS) que funcione de forma distribuída, utilizando um modelo cliente-servidor, sendo que os agentes são responsáveis pela coleta de um conjunto de métricas, sendo elas reportadas a um servidor centralizado responsável por armazenar a informação recebida.

O sistema de monitorização apresentado neste trabalho é inspirado no protocolo "NetFlow". No entanto, será implementado com base em dois protocolos aplicacionais distintos, denominados NetTask e AlertFlow.

Arquitetura da solução



Especificação dos protocolos propostos

Formato das mensagens protocolares

Protocolo NetTask (UDP)

Handshake (registo de agente):

SEQ: 1	ACK: 0	FLAGS: SYN	AGENT_ID: Agent01

Resposta do Handshake:

SEQ:2	ACK:2	FLAGS: SYN-ACK

Diagrama de sequência da troca de mensagens

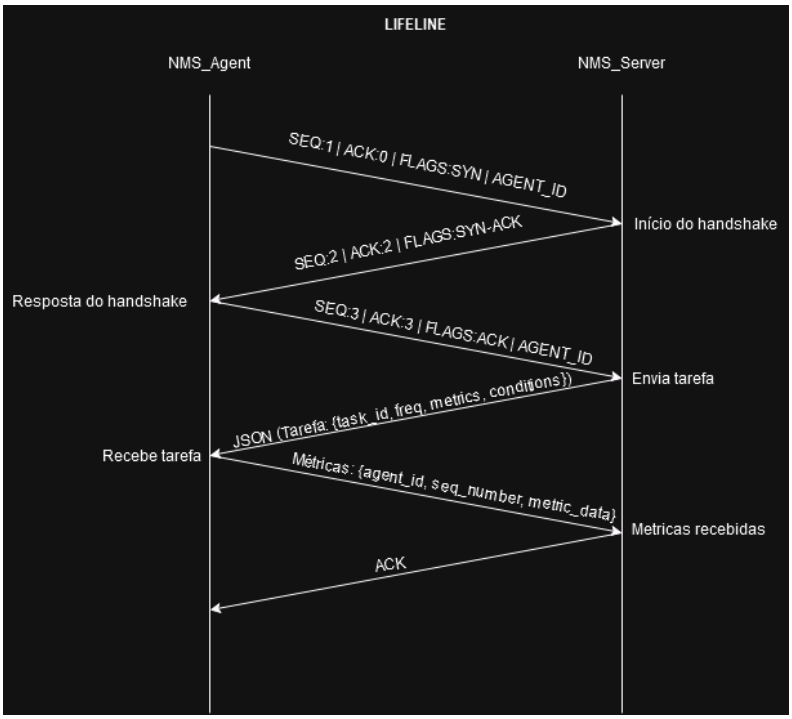


Figura 1 - Lifeline

Implementação

NMS_Server

A classe **NMS_Server** é responsável por implementar um servidor para gerir a rede (Network Management Server), cuja principal função é interagir com os agentes distribuindo tarefas, processar métricas enviadas por eles e gerenciar a comunicação por meio de protocolos UDP e TCP.

A estrutura desta classe esta dividida em diferentes partes que desempenham funções especificas, sendo inicialmente configurados parâmetros como endereço de *host* e número das portas utilizadas para os protocolos udp e tcp.

Para armazenar as informações foram utilizadas coleções concorrentes como o ConcurrentHashMap. Três estruturas principais são utilizadas:

- "***agents***", que mapeia os identificadores dos agentes conectados para informações como endereço e tempo da última comunicação;
- "***metricsData***", que armazena os dados de métricas enviados pelos agentes
- "***tasks***", que mantém as tarefas atribuídas a cada agente, incluindo informações como frequência de envio e condições para geração de alertas.

Uma parte crucial da funcionalidade do servidor está na comunicação via UDP. Por meio do método "udpListener", o servidor escuta mensagens enviadas pelos agentes. Estas mensagens podem incluir sinalizações de conexão, como FLAGS: SYN e ACK, ou dados mais complexos, como métricas de desempenho. Ao receber uma mensagem de conexão, os métodos "handleSynMessage" e "handleAckMessage" gerenciam o processo de *handshake*, registando novos agentes ou atualizando informações de agentes já conectados. No caso de receber métricas de desempenho o método "processMetricMessage" processa os dados recebidos e guarda-os num arquivo CSV.

A comunicação via TCP é utilizada principalmente para lidar com alertas enviados pelos agentes. O método "tcpMainThread" é responsável por escutar conexões de entrada, enquanto o "handleTcpConnection" processa as mensagens específicas, como os alertas. Esses alertas são então registados em arquivos CSV dedicados, possibilitando que sejam analisados posteriormente.

O servidor também desempenha um papel ativo na gestão de tarefas carregadas a partir de um ficheiro JSON, utilizando o método "loadTasks". Cada tarefa contém informações detalhadas, como a frequência de envio e as condições de alertas, que são atribuídas a agentes específicos. O método "sendTaskToAgent" é utilizado para enviar essas tarefas aos agentes registados por meio de mensagens UDP.

NMS_Agent

A classe **NMS_Agent** é responsável pela implementação de um agente de monitoração que interage com o servidor central (**NMS_Server**) para realizar tarefas de coleta de métricas de rede (NetTask) e envio de alertas críticos (AlertFlow).

O método "receiveTask" processa as tarefas recebidas, que indicam um conjunto de métricas a serem coletadas e condições de alerta. Estas métricas são coletadas no método "collectMetrics" e enviadas ao servidor periodicamente através do método "sendMetrics", que encapsula os dados em formato JSON e os transmite via protocolo UDP. Quando os limites definidos são ultrapassados o método "sendAlert" é acionado para enviar uma notificação ao servidor via TCP.

De forma geral, a classe **NMS_Agent** processa as tarefas recebidas, coleta as métricas, envia-as ao servidor em intervalos regulares e envia alertas quando necessário.

Testes e Resultados

Para testar o sistema, foram realizados os seguintes testes em um ambiente CORE no XubunCORE 7.5, uma vez que este permite configurar topologias de rede, ajustar parâmetros como largura de banda, latência e perda de pacotes, além de realizar o monitoramento detalhado das interações entre os nós.

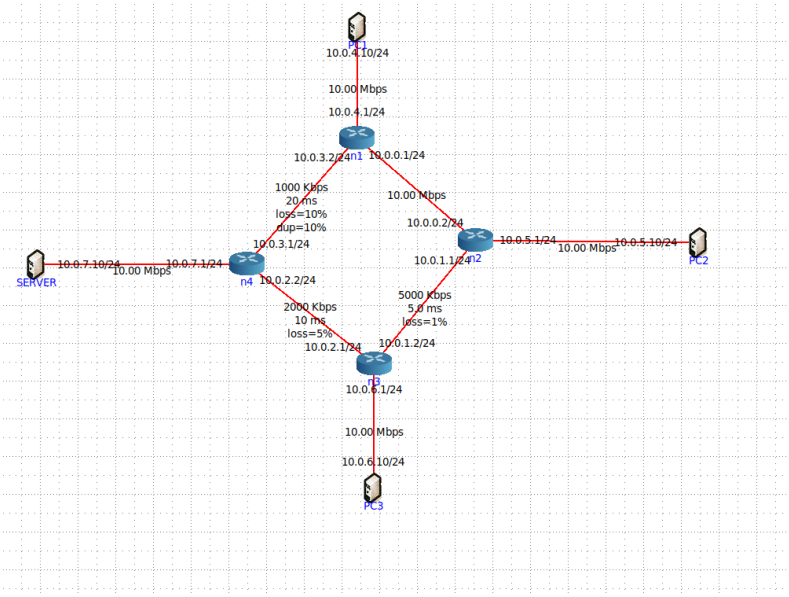


Figura 2 - Topologia de testes

Handshake

```
root@PC4:/tmp/pycore.43271/PC4.conf# cd PorjetoCC/
root@PC4:/tmp/pycore.43271/PC4.conf/PorjetoCC# java -cp gson-2.11.0.jar NMS_Server.java

[IPERF3] Iperf3 server started.

[SERVER] Tasks loaded successfully.

[UDP] Server waiting for messages...

[TCP] Server waiting for connections...
```

Figura 3 - Inicializador do Servidor

Figura 4 - Handshake do Agente

```
[UDP] Received: SEQ:1|ACK:0|FLAGS:SYN|AGENT_ID:n1 from /10.0.3.2:56546
[HANDSHAKE] Sent: SEQ:2|ACK:2|FLAGS:SYN-ACK

[UDP] Received: SEQ:3|ACK:3|FLAGS:ACK|AGENT_ID:n1 from /10.0.3.2:56546
[HANDSHAKE] Handshake completed for n1

[SERVER] Task sent to n1

[TCP] Connection received from /10.0.3.2:60782
```

Figura 5 - Handshake do Servidor

De forma a validar a comunicação entre o servidor e o agente, foram realizados os testes das figuras 3, 4 e 5, cujos resultados confirmam que o sistema de monitorização e comunicação entre agente e servidor cumpre os requisitos especificados. Os handshakes, tanto TCP quanto UDP, foram validados, e o registo de tarefas funcionou conforme planejado.

Largura de banda

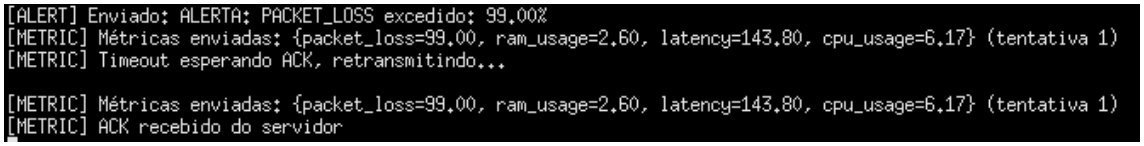


Figura 6 - Largura de banda 10kbps enviada

Figura 7 - Largura de banda 10kbps recebida

No teste da Figura 5, o agente está a enviar métricas ao servidor com uma largura de banda limitada a 10 kbps. Devido à largura de banda ser muito baixa, a latência aumenta, ocorrendo uma perda significativa de pacotes, sendo necessário que o sistema retransmita os dados para garantir a entrega final.

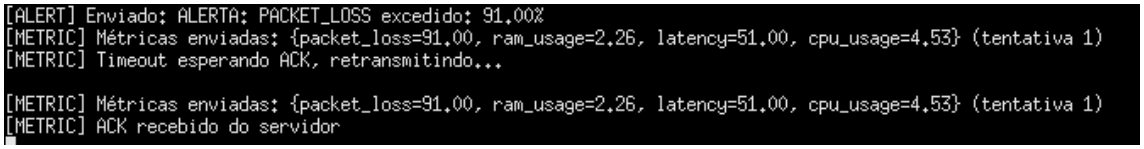


Figura 8 - Largura de banda 100kbps enviada

Figura 9 - Largura de banda 100kbps recebida

No teste da imagem 7, o agente está a enviar métricas ao servidor sob uma largura de banda limitada a 100kbps sendo que apesar do packet_loss ainda ser significativo, a latência é reduzida em relação ao teste anterior(10kbps).

Perda de pacotes

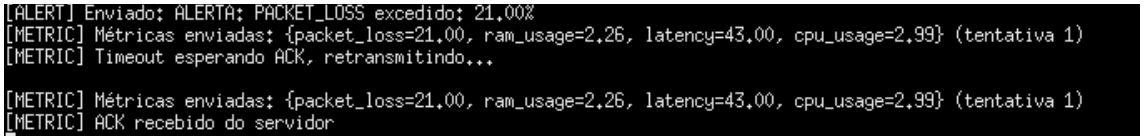


Figura 10 - Alerta enviado

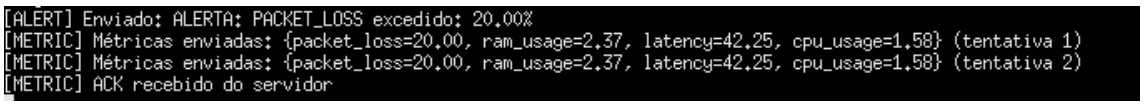


Figura 11 - Alerta enviado

Figura 12 - Alerta recebido

Figura 13 - Alerta recebido

No contexto dos testes realizados, as imagens mostram o comportamento do sistema na deteção, envio e receção de métricas e alertas relacionados à perda de pacotes (*Packet Loss*). O sistema foi testado para diferentes limites de perda de pacotes (21% e 20%), com o objetivo de validar sua capacidade de retransmissão em caso de falhas e assegurar a entrega dos dados ao servidor.

Os testes demonstraram que o sistema é resiliente a falhas temporárias de comunicação, utilizando mecanismos de retransmissão para garantir a entrega dos dados. Além disso, a deteção e envio de alertas foram realizados corretamente.

Partilha de métricas

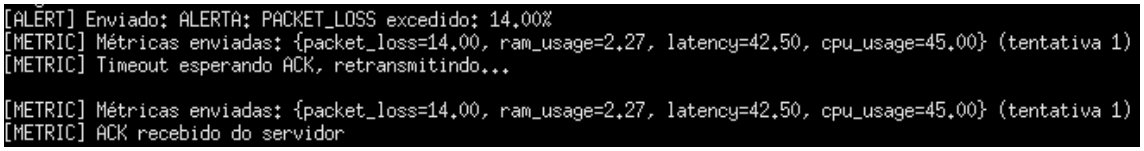


Figura 14 - Métricas enviadas

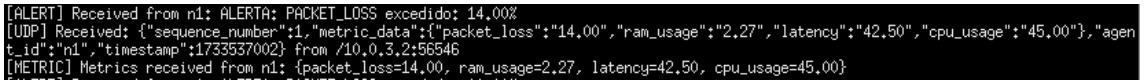


Figura 15 - Métricas recebidas

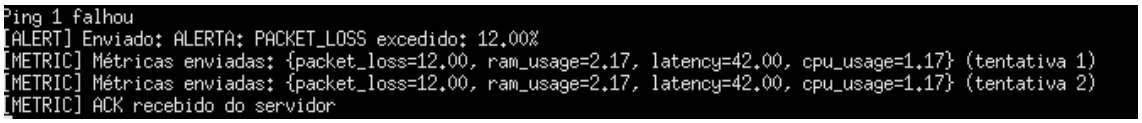


Figura 16 - Métricas enviadas

Figura 17 - Métricas recebidas

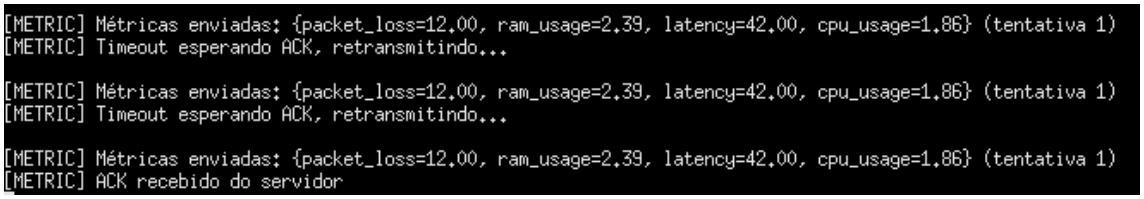


Figura 18 - Métricas enviadas

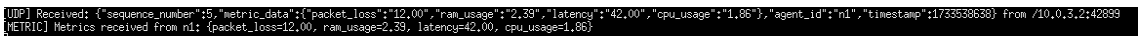


Figura 19 - Métricas recebidas

As Figuras 14 a 19 mostram o processo de partilha de métricas entre os agentes e o servidor no contexto dos testes realizados. Este processo é essencial para a monitorização contínua do desempenho e estabilidade da rede, especialmente em situações de alerta, como a deteção de perda de pacotes.

Os testes demonstram que o sistema é capaz de lidar com falhas temporárias de comunicação, utilizando retransmissões para garantir a entrega dos dados. O comportamento exibido nas Figuras 14 a 19 reflete a resiliência e fiabilidade do sistema em cenários adversos, garantindo que métricas e alertas sejam processados e registados adequadamente no servidor.

Storage(CSV)

```
Agent ID, Timestamp, Sequence Number, Metric = Value
n3, 1733540465, 0, packet_loss = 5.30
n3, 1733540465, 0, ram_usage = 2.05
n3, 1733540465, 0, latency = 23.80
n2, 1733540470, 1, latency = 52.00
n2, 1733540470, 1, cpu_usage = 15.19
n1, 1733540479, 1, packet_loss = 11.00
n1, 1733540479, 1, ram_usage = 2.43
n1, 1733540479, 1, latency = 42.80
n1, 1733540479, 1, cpu_usage = 24.23
```

Figura 20 - Metricas

```
Agent ID, Timestamp, Alert Message
n1, "1733540447631", ALERTA: PACKET_LOSS excedido: 11.00%
n3, "1733540465326", ALERTA: PACKET_LOSS excedido: 5.30%
n1, "1733540479933", ALERTA: PACKET_LOSS excedido: 11.00%
n1, "1733540511976", ALERTA: PACKET_LOSS excedido: 11.00%
n3, "1733540536639", ALERTA: PACKET_LOSS excedido: 7.30%
n3, "1733540579423", ALERTA: PACKET_LOSS excedido: 7.00%
n3, "1733540693178", ALERTA: PACKET_LOSS excedido: 5.10%
```

Figura 21 – Altertas

Nas figuras 15 e 16 é apresentada a estrutura com que os dados das métricas coletadas e dos alertas gerados estão a ser armazenados nos ficheiros CSV. A figura 15 mostra o formato de armazenamento das métricas, que incluem informações como o ID do agente, *timestamp*, número de sequência e os valores de métricas monitoradas, como perda de pacotes (*packet_loss*), uso de RAM (*ram_usage*), latência (*latency*) e uso de CPU (*cpu_usage*). Já a figura 16 apresenta o registo dos alertas gerados, onde são armazenados o ID do agente, o *timestamp* e a mensagem correspondente ao alerta, indicando que um determinado limite foi excedido. Esta estrutura facilita a organização e a análise dos dados, permitindo identificar rapidamente problemas e realizar diagnósticos.

Conclusão e Trabalho Futuro

O desenvolvimento deste projeto permitiu-nos consolidar e aplicar conceitos abordados durante as aulas teóricas exigindo uma boa compreensão de protocolos da camada de transporte (TCP e UDP) e sockets.

Além disso, o trabalho aprofundou o uso da linguagem Java em um contexto distribuído, demonstrando sua flexibilidade na manipulação de comunicações em rede e processamento paralelo. Este projeto mostrou a importância da monitorização contínua para garantir o desempenho e a estabilidade das redes.

Como trabalho futuro, os protocolos NetTask e AlertFlow podem ser aprimorados para tornar o sistema mais eficiente e seguro, garantindo uma melhor performance e confiabilidade.